SONY

MSX 2+

MSX-BASIC Version 3.0

解説書

HIT BIT

この本でMSX-BASICの勉強をされるかたへ

この本は「入門編」と「応用編」の2部からできています。「入門編」は，BASICはまったく始めて，というかたも最初から読んで実際にコンピューターを扱っていくうちにBASICとはどんなものか，BASICでプログラムを組むとはどういうことなのかが理解できるように書かれています。また，練習をとおしてBASICでプログラムを作るのにぜひ必要な基本の命令語を勉強できるようにしています。
MSX-BASICについての知識があるかたは，「入門編」はとばして「応用編」から読んでいただいてもかまいません。
また，MSX-BASICは始めてだけれど，他のBASICをかじったことのあるかた，あるいはかなり知識のあるかたは，「入門編」にざっと目をとおして，MSX-BASICの基本的な命令語にはどんなものがあるのかを確認してから「応用編」にすすむと良いでしょう。

MSX-BASIC Version 3.0について

MSX-BASIC Version 3.0は，今までのMSXホームコンピューターに搭載されていたMSX-BASIC Version 2.0の，グラフィック処理機能と漢字処理機能を拡張したものです。さらに，MSXの拡張仕様であるFM音源（MSX-MUSIC）搭載のコンピューターでは，FM音源用の拡張コマンドも使用できるようになりました。
MSX-BASIC Version 3.0は，MSX-BASIC Version 2.0の命令語，関数，機能をすべて含んでいます。したがってMSX-BASIC Version 2.0で組まれたプログラムはMSX-BASIC Version 3.0を搭載したコンピューターでも利用することができます。
この本では，MSX-BASIC Version 3.0の使いかたを説明しますが，文中では，とくにMSX-BASIC Version 3.0だけにかかわる機能などの説明の際はMSX_{2+}-BASICという呼びかたで統一します。それ以外の場合は，単にMSX-BASICあるいはBASICと呼ぶことにします。

MSXはアスキーの商標です。

目 次

入門編

第1章　プログラムってなに？

コンピューターに命令してみよう ……10
準備はいいですか ……10
BASICスタート ……11
命令はキーボードから ……11
計算してみよう　PRINT 式 ……12
変数に数値を入れる　LET ……13
文字を表示させる　PRINT〝文字列〟 ……16
文字型変数には$マーク ……17
プログラムにしてみよう ……19
直接モードとプログラムモード ……19
たった1行のプログラム ……19
プログラムの確認　LIST ……20
プログラムの実行　RUN ……21
記憶を消す　NEW ……22
変数へ入れる値をキーボードから　INPUT ……23
PRINT命令を使いこなそう ……25
プログラムの一部を消す　DELETE ……28
画面の表示を消す　CLS ……28
ジャンプする　GOTO ……29

第2章　もっとコンピューターらしく

ループからの脱出 ……32
「もし〜なら」という条件判断　IF-THEN ……32
プログラムを終了させる　END ……34
条件式 ……34
行番号の整理　RENUM ……35
繰り返しのプロフェッショナル ……36
回数を決めてループ　FOR-NEXT ……36

データがいっぱい ……39
変数に値を入れるもうひとつの方法　READ-DATA ……39
プログラムをディスクに記録するには ……41
ディスクのフォーマット　CALL FORMAT ……41
プログラムのセーブ　SAVE ……43
セーブされたか確認　FILES ……44
プログラムのロード　LOAD ……44
不要なプログラムを消す　KILL ……45
プログラムをコピーする　COPY ……45

第3章　配列変数を使おう

配列変数を使ったプログラム ……50
配列変数の使いかた　DIM ……50

応用編

第4章　画面の構成とグラフィック

画面のモード ……54
画面の構成 ……54
画面モードの設定　SCREEN ……54
キャラクターモード ……57
1行の文字数の指定　WIDTH ……57
グラフィックモードと座標 ……59
マルチカラーモード（SCREEN 3） ……60
STEPの指定 ……62
カラーの指定 ……63
カラーコードとパレット機能 ……63
パレット機能の使いかた ……65
パレットの設定　COLOR ……66
SCREEN 6モードとパレット機能 ……69
SCREEN 8モードとカラー ……69

SCREEN 2とSCREEN 4モードの色バケについて……70
SCREEN 10、SCREEN 11、SCREEN 12モードとカラー……71
初期状態に戻す COLOR……82
ページの切り換え……85
グラフィックモードの画面とページ……85
ページ切り換えによる効果……87
ページを切り換えるには SET PAGE……88
図形データの転送……90
転送とは……90
画面間の転送 COPY その1……91
画面と内部メモリー間の転送 COPY その2……94
画面とフロッピーディスク間の転送 COPY その3……99
フロッピーディスクとメモリー間の転送 COPY その4……101
ロジカルオペレーション……101
SCREEN文でできること……107
SCREEN文で設定できること……107
キークリックスイッチ，ボーレート，プリンタータイプ……107
インターレースモード……109

第5章 スプライトを動かそう

スプライトの定義と動かしかた……112
スプライトとは……112
スプライトパターンの定義 SPRITE$変数……114
スプライトパターンの表示 PUT SPRITE……118
スプライトを動かす……120
機能強化スプライトを使う……122
スプライト機能の強化……122
スプライトの色を変える COLOR SPRITE……122
1ラインごとに色を指定する COLOR SPRITE$……124
スプライトパターン定義のテクニック……128

第6章　日本語を使おう

日本語入力モードにする……132
日本語入力モードとは……132
日本語入力モードにするには　CALL KANJI……133
英数字入力モードに戻すには　CALL ANK……134
日本語を使ってプログラムを作る……135
こんなプログラムを作ってみよう……135
連文節変換によりプログラムを入力する……136

第7章　音楽を演奏しよう

メロディーの演奏……144
MSXと音楽演奏　PLAY……144
音の高さを決める　A～Gサブコマンド、Oサブコマンド……145
音の長さを決める　Lサブコマンド……146
演奏の速さを決める　Tサブコマンド……147
音色を変える　Sサブコマンド，Mサブコマンド……148
和音を演奏する……150
効果音をつくる……152
PSGとレジスター……152
効果音を作る　SOUND……152
自動車とSLを走らせてみる……153
FM音源を使う……157
FM音源（MSX-MUSIC）について……157
使うチャンネルを決める　CALL MUSIC……157
各チャンネルの音色を決める　CALL VOICE……160
FM音源で演奏する　PLAY……162

第 8 章　関数に強くなろう

数値型の関数 …… 166
　関数とは …… 166
　数値型の関数 …… 166
　平方根を求める　SQR(X) …… 167
　三角関数　SIN(X) …… 169
　絶対値を返す関数　ABS(X) …… 170
　ランダム関数　RND(X) …… 171
　整数に直す関数とRND(X)　INT(X) …… 171
文字型の関数 …… 176
　文字型関数とは …… 176
　必要なだけスペースを　SPACE$(N),SPC(N) …… 176
　文字列の加工　LEFT$(X$,N),MID$(X$, M, N), RIGHT$(X$, N) …… 177
　日本語文字列の処理　CALL KMID (A$, X$, M, N) など …… 179
数値型・文字型変換の関数 …… 181
　数値型・文字型変換関数のいろいろ …… 181
　数字の型を変える　VAL(X$), STR$(X) …… 181
　文字コードと関数　ASC(X$),CHR$(X) …… 183
　文字列の長さを返す　LEN(X$) …… 184
データ入力の関数 …… 185
　データ入力関数の働き …… 185
　キーボードからデータ入力　INKEY$ …… 187
　カーソルキーの状態を入力　STICK(N) …… 189

第 9 章　割り込み

割り込みをかけるには …… 192
　割り込みとは …… 192
　MSX-BASICで使える割り込み …… 192
　割り込みをかけるには …… 193

割り込みを使ったプログラム ……194
ファンクションキーによる割り込みプログラム ……194
割り込みを無効にする KEY(N)OFF ……195
割り込みの保留 ……196
割り込み処理ルーチンでさらに割り込みを有効にする KEY(N)ON ……197
プログラムで割り込みを保留にする KEY(N)STOP ……198
スプライト衝突による割り込み ……200

第10章 ファイル処理

ファイルとファイルデバイス ……204
ファイルとファイル名 ……204
ファイルデバイスとデバイス名 ……205
ファイル名のルールとタイプ名 ……208
プログラムファイルとデータファイル ……209
プログラムファイルの操作 ……210
ファイル管理 ……216
オートスタートのプログラムファイル ……216
シーケンシャルファイルの操作 ……217
シーケンシャルファイルとランダムアクセスファイル ……217
シーケンシャルファイルへの書き込み OPEN FOR OUTPUT ……218
シーケンシャルファイルからの読み出し OPEN FOR INPUT ……221
データの追加 OPEN FOR APPEND ……224
グラフィック画面に文字を表示する ……226
1度に開くことのできるファイルの数 MAXFILES ……227
ランダムアクセスファイルの操作 ……228
ランダムアクセスファイルとは ……228
ランダムアクセスファイルへの書き込み ……230
ランダムアクセスファイルからの読み出し ……235

第11章　機械語サブルーチン

機械語サブルーチンの作成と実行 ……238

機械語サブルーチン ……238

サブルーチン領域の設定と先頭番地の定義　CLEAR, DEFUSR ……238

機械語サブルーチンの書き込み　POKE ……239

機械語サブルーチンの実行　USR ……240

機械語プログラムのセーブ　BSAVE ……245

機械語プログラムのロード　BLOAD ……245

さくいん

BASICの命令，文，関数，エラーメッセージなど ……246

用語など ……249

こんなときには ……251

付録

MSX2+-BASICリファレンスチャート ……255

入門編

第1章
プログラムってなに？

この本を読む前に，次の文を実行してください。

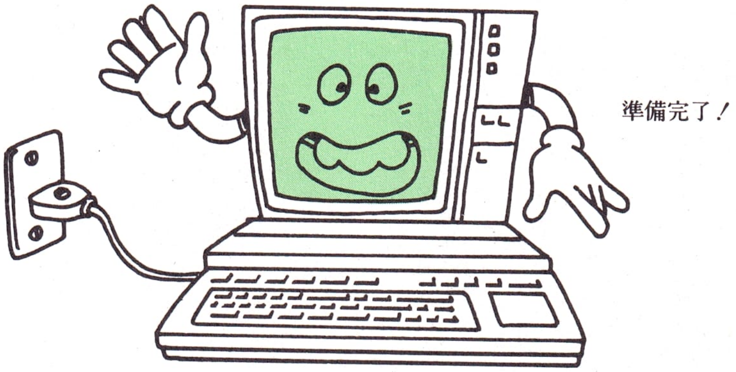
```
SET TITLE "",1
SET PROMPT "Ok"
SET ADJUST (0,0)
SET BEEP 1,3
SCREEN 1,,1,1,0,0
KEY ON
WIDTH 29
COLOR 15,4,7
```

コンピューターに命令してみよう

- 命令の与えかた
- MSX-BASICとは
- 計算させる——PRINT 式
- 変数とLET命令
- 文字を表示させる——PRINT "文字列"
- 文字型変数

準備はいいですか

では，いよいよ実際にコンピューターの操作をしながらBASICの勉強開始です。もう，コンピューターとモニターテレビなどとの接続はすんでいますか。まだなら，「取扱説明書」をみて正しく接続してください。この本に書いてあることを練習するためには，コンピューターとモニターテレビは最低限必要です。



ひとこと

モニターテレビとして，スピーカーのついていないカラーモニターを使う場合は，モニターに外部スピーカーを接続しておいてください。「音」もコンピューターを操作していく上でとても大事な働きをしますから。

BASICスタート

接続ができたら，電源を入れましょう。やはり，「取扱説明書」をみて操作してください。BASICのスタートの方法も「取扱説明書」に書いてありますね。それに従って，BASICをスタートさせると，モニターテレビは次のようになるはずです。

```
MSX BASIC version 3.0
Copyright 1988 by Microsoft
xxxxx Bytes free
Disk BASIC version 1.0
Ok
```

下から2行目にOkと表示され，その下に四角いマークがあらわれていれば，BASICの準備がととのったということです。

モニターテレビが上のようにならなかったときは，次の文を入力してください。上の画面と同じ状態になります。

```
SET TITLE "",1
SET PROMPT "Ok"
SET ADJUST (0,0)
SET BEEP 1,3
SCREEN 1,,1,1,0,0
KEY ON
WIDTH 29
COLOR 15,4,7
```

次に電源を入れるときは，上の画面からスタートします。

命令はキーボードから

「コンピューターに命令してみよう」というのがこの章のタイトルでした。では，コンピューターには，いったいどうやってあなたからの命令を伝えるのでしょうか。

コンピューターの命令の入り口ともいえるのが，コンピューターの上にたくさん並んだ[A][B][C]や[1][2][3]などのキーの集りです。このキーの集りのことをキーボードと呼びます。あなたは，このキーボードをたたいてコンピューターにいろいろな命令を与えるのです。コンピューターに命令するときに使うことばにはいろいろありますが，この本ではこれからBASICということばを使って命令することになります。

計算してみよう　PRINT　式

BASICを使った最初の命令として，10＋5の足し算をさせてみましょう。次のようにキーを押してください。

[P][R][I][N][T][　][1][0][＋][5][RETURN]

画面は次のようになります。

```
PRINT 10+5
 15
Ok
```

コンピューターは，PRINT　10＋5という命令に従って，10＋5の計算を行ってその答の15を表示してくれました。

PRINT　式（プリント）

PRINTという命令は画面に表示しなさいというBASICの命令のひとつ，つまりコンピューターに分かる命令のひとつなのです。そして，PRINTの次に計算式を書くと，「その式を計算して答を表示しなさい」という意味の命令になります。

計算あれこれ

「10＋5」は，簡単な足し算でしたが，もっと難しい計算はどうでしょうか。まず，計算の記号ですが，BASICでは次のような記号を使います。

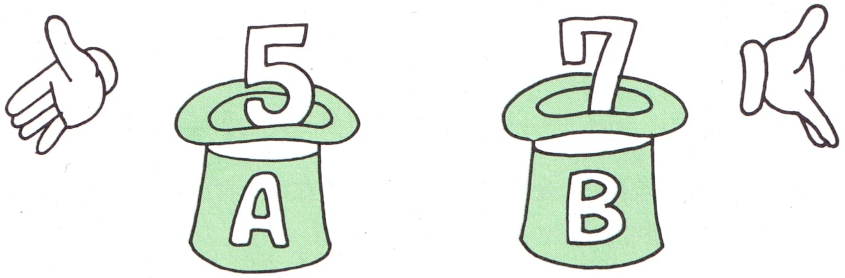
	記号	例	意味
足し算	＋	PRINT　123＋234	123＋234
引き算	－	PRINT　300－125	300－125
かけ算	＊	PRINT　9＊8	9×8
割り算	／	PRINT　72／36	72÷36
累　乗	^	PRINT　5^3	5^3

小カッコ（　）や中カッコ｛　｝，大カッコ［　］は，みなコンピューターでは（　）を使います。この場合，内側のカッコから順に計算されます。

変数に数値を入れる　LET

数をしまう

今,「A」と名の付けられた帽子の中に5という数が,「B」という帽子の中には7という数が入っているとします。



A+Bはいくつになるでしょうか。5+7=12ですね。ではA×Bは？　35です。

これと同じことをコンピューターでやってみましょう。まず,「A」に5,「B」に7を入れるところから始めます。次のように命令してください。

L E T ␣ A = 5 RETURN

L E T ␣ B = 7 RETURN

```
LET A=5
Ok
LET B=7
Ok
■
```

コンピューターは,ただ「Ok」と答えるだけですが,コンピューターの内部では,ちょうど帽子の中に数を入れたのと同じようなことが起こっています。

LET命令

LET(レット)　変数=数値

LETは,ある変数にある数をしまうという命令です。

今コンピューターに与えた命令の中で,AやBのような文字が帽子に当たります。そして,このように数をしまうために使われるAやBなどの文字を変数と呼びます。

変数を使った計算

続いて，次のように命令してみましょう。

```
PRINT A+B
 12
Ok
PRINT A*B
 35
Ok
```

「PRINT　式」命令の「式」の部分に変数どうしの計算式を書いたら，変数に入っている数に従って正しい答を出してくれました。

さて，ここで変数を使うときのルールをまとめておきましょう。

変数を使うときのルール

■変数の長さは何文字でもかまいませんが，コンピューターは初めの2字だけを区別します。したがって，

ABC，ABCD，ABD，AB1，AB2

などは皆同じ変数になります。

■変数の最初の文字には数字や記号は使えません。

A1，C3は変数として使えますが，1AB，＊ABなどは使えません。

■BASICの命令や関数として使われることばは変数として使えません。また，変数の一部分にこれらのことばは含んでもいけません。

PRINTやLET，それにPRINTA，ALETなどはすべて変数として使えません。

LET命令の秘密——LETは省略できる

LET命令を書くときは，「LET」は省略しても良いことになっています。

では，「LETを省略したLET命令」と，新しい変数を使ってもう少し練習してみましょう。

```
PRINT AB      } ABという変数にはまだ何も入れて
 0            } いません。このとき中身は0です。
Ok
AB=100 ——— 変数ABに100を入れました。
              (LET　AB＝100と同じです。)
Ok
CD=200 ——— 変数CDに200を入れました。
Ok
X=AB+CD ——— 変数XにABとCDの足し算の結果
              (300) を入れました。
Ok
PRINT X ——— Xを表示しました。
 300
Ok
PRINT ABC     } ABCという変数はABと同じとみな
 100          } されます。
Ok              したがって中身は100です。
```

文字を表示させる　PRINT "文字列"

こんどは，画面上に好きな文字を表示させる命令です。この場合も，やはりPRINT命令が使えます。さっそく練習してみましょう。次の命令をためしてみてください。

[P][R][I][N][T][　][″][S][O][N][Y][″][RETURN]

[″]はクォーテーションマークで，[SHIFT]キーを押しながら[2]を押します。

```
PRINT "SONY"
SONY
Ok
```

このように，表示させたい文字を"　"で囲んでPRINTに続ければ，その文字が表示されます。"　"で囲まれた文字の集りは，文字列と呼びます。

PRINT "文字列"

"　"の中をいろいろ変えて練習してみてください。たとえば次のように。

```
PRINT "3+5"
3+5
Ok
```

3+5は計算ですが，それを"　"で囲んであるので，全体をサン，タス，ゴという3つの文字として扱っています。"　"をはずしたらどうなるかはもう分かりますね。3+5が計算され，結果の8が表示されるはずです。PRINT "3+5"の場合，3+5は式(数値）ではなく，文字列として扱われたのです。

コンピューターでは，数字は，数値あるいは文字のどちらとしても扱うことができます。これは，とても大切なことなので，覚えておいてください。

文字だって足し算できる——文字列の加工

文字の足し算というのは，文字どうしをつなぎ合わせることです。次の例をためしてみましょう。

```
PRINT "SO"+"NY"
SONY
Ok
PRINT "HOME "+"COMPUTER"
HOME COMPUTER
Ok
```

最初の例では，文字列”SO”と”NY”がつながって”SONY”になりました。次の例では，”HOME”のEの次に1文字分の空白があります。（これは，スペースバーを押せば良いですね。）そこで，文字列どうしの足し算の結果でも”E”と”C”の間に1文字分の空白ができています。空白（スペース）も，1文字分として扱われているのですね。

さて，文字列どうし，こんな風につなぎ合わせができるとなると，文字をしまっておく帽子（変数）も欲しくなってきませんか。

文字型変数には$マーク

AやB，CDなどの変数には数をしまっておくことができました。同じように，文字列をしまっておく変数というものも使うことができます。文字列をしまう変数のことを文字型変数といって，数をしまう変数と区別します。数をしまう変数のことは数値型変数と呼びます。
文字型変数は，A$，B$，CD$などのように後ろに$をつけます。$がついた変数には，文字列しか入れることができません。

```
LET A$="SONY"
Ok
PRINT A$
SONY
Ok
```

LET命令を使って，文字型変数A$に文字列”SONY”を入れました。もちろん，この場合もLETは省略してかまいません。また，文字型変数に文字列を入れるときは，文字列を必ず”　”で囲まなければなりません。（数字も，文字として扱えば入れることができます。）

```
B$="123"
Ok
PRINT B$
123
Ok
C$=123
Type mismatch
```

C$という文字型変数に123という数値を入れようとしたところ,”Type mismatch(タイプミスマッチ)”というメッセージが返ってきました。”Type mismatch”というのは「型が合っていません」という意味です。文字型の変数に数値を入れようとしたので,型がちがいますよ,とコンピューターが教えてくれたのです。

エラーメッセージについて

コンピューターは,あなたが間違った命令を与えると,それはどんな間違いなのかも分かるようにメッセージを返してくれます。このようなメッセージのことを**エラーメッセージ**といいます。「命令の意味が分かりません」という意味の

```
Syntax error
```

や,「型が合っていません」という意味の

```
Type mismatch
```

などがそうです。

プログラムにしてみよう

- プログラムの入力
- プログラムの確認――LIST
- プログラムの実行――RUN
- 記憶しているプログラムを消す――NEW
- 変数の値をキーボードから代入――INPUT
- PRINT命令あれこれ
- 行を消す――DELETE
- 画面の表示を消す――CLS
- ジャンプする――GOTO

直接モードとプログラムモード

今までの練習のように，ひとつの命令を RETURN キーを押して1回実行しておしまい，というコンピューターの使い方を，直接モードで使う，といいます。コンピューターには，もうひとつの使い方があり，それをプログラムモードといいます。そして，プログラムモードこそコンピューターをコンピューターらしく使う方法なのです。

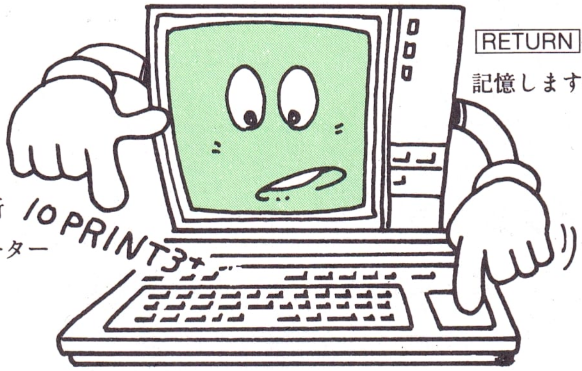
たった1行のプログラム

では，これからプログラムモードでプログラムを作ってみましょう。プログラムモードといっても，別に何か特別な操作をするわけではありません。次のようにキーボードを押してみてください。

1 0 ␣ P R I N T ␣ 3 ＋ 5 RETURN

```
10 PRINT 3+5
■
```

PRINT 3+5という命令は,3+5という式を計算して,その答(8)を表示させるものでした。でも,今回は最後に[RETURN]キーを押しても答は表示されません。なぜでしょうか。それはPRINT 3+5という命令の前につけた10という番号のせいなのです。コンピューターは,命令の前に番号がつけられていると,最後の[RETURN]キーで命令を実行する代わりにとりあえずそれを記憶します。この番号のことを,行番号と呼びます。



この,コンピューターが記憶した,行番号つきの命令がプログラムと呼ばれるものです。

プログラムの確認　LIST

コンピューターは今,

```
10 PRINT 3+5
```

というプログラムを記憶しています。それを確かめるための命令として,

[LIST(リスト)]

というものがあります。

[L][I][S][T][RETURN]

とキーボードを押してみてください。

```
LIST
10 PRINT 3+5
Ok
```

LISTは,プログラムリストを表示しなさい,という意味の命令です。LIST命令は直接モードで使ったことはお分かりですね。

プログラムの実行　RUN

直接モードでは，命令を実行させるには RETURN キーを押しました。プログラムを実行させるときは，RUN命令を使います。

RUN（ラン）

R U N RETURN

と押してみましょう。

```
RUN
 8
Ok
```

今記憶しているプログラム

```
10 PRINT 3+5
```

がここで初めて実行され，答の8が表示されました。

もう一度RUN命令を与えると，やはり8が表示されます。何回繰り返しても同じことです。

プログラムとは？

もう一度，

```
PRINT 3+5
```

という命令を直接モードで実行した場合とプログラムにした場合のことを比べてみましょう。

	直接モードでは	プログラムモードでは
行番号	つけない	つける
実行方法	RETURN キーを押す	RUN命令を与える
命令の記憶	記憶されない	記憶される
実行回数	1回だけしかできない	何回でもできる

このように，プログラムとは，行番号をつけたためにコンピューターがそれを記憶し，RUN命令によって何回でも実行できるものです。

そして，プログラムを記憶できるということこそが，コンピューターのコンピューターたるゆえんといってもかまわないくらい，プログラムとコンピューターは切っても切り離せないものなのです。

記憶を消す　NEW

これから，別のプログラムを作ってコンピューターに記憶させたいのですが，今コンピューターは，

```
10 PRINT 3+5
```

というプログラムを記憶しているはずです。そこで，新しいプログラムを記憶させる前にNEWという命令を実行します。

ニュー NEW

NEW命令は，今記憶しているプログラムをすべて消す命令です。

[N][E][W][RETURN]

と押してください。そして，LIST命令で，プログラムが消えていることを確かめてみましょう。

```
NEW
Ok
LIST
Ok
```

もう，プログラムは表示されませんね。

記憶を消す方法

記憶されたプログラムは，NEW命令で消すことができますが，それ以外に次の場合も記憶は消えてしまいます。

- [RESET] ボタンを押したとき
- コンピューターの電源を切ったとき

ですから，プログラムを作っている最中などに，まちがって [RESET] ボタンや電源スイッチを押さないよう注意しなければなりません。

変数へ入れる値をキーボードから　INPUT

こんどは，実行するたびに変数A，Bに違った値を入れることができるプログラムを作ってみたいと思います。そうすれば1回ごとに異った値の足し算ができるようになります。そのためにはLET命令に代わる命令が必要です。

ここで，新しくINPUT命令を覚えましょう。

INPUT（インプット）　変数名

INPUTは，キーボードから入力された値を変数に入れなさいという命令です。

```
10 INPUT A
20 INPUT B
30 PRINT A+B
```

プログラムを実行すると

プログラムの入力がすんだらさっそくRUN命令で実行です。

```
RUN
? ■
```

今，コンピューターは，行10のINPUT Aという命令により，あなたがキーボードを使って変数Aに入れる値を指示してくれるのを待っているのです。そこで，好きな値をキーボードから入れることにしましょう。何けたでもかまいませんが，25を入れることにしましょうか。

[2][5][RETURN]

と押してください。

```
RUN
? 25
? ■
```

すると，再び　?があらわれました。こんどは，行20が実行され，変数Bには何を入れますか，とたずねているのですね。変数Bには75を入れてみましょうか。

[7][5][RETURN]

で良いですね。

```
RUN
? 25
? 75
 100
Ok
```

変数Aに25，変数Bに75が入ったところで，行30が実行されました。A＋Bの結果の100が表示されましたね。

もっと親切になれるINPUT命令

このプログラム，実行するといきなり？が表示されます。プログラムを作った本人なら，今は変数Aの値を入れるんだ，ということが分かりますが，別の人がこのプログラムを実行したらいったいどうすれば良いのか分からなくなってしまいます。そこで，INPUT命令を次のように変更してみましょう。

```
10 INPUT "A=";A
20 INPUT "B=";B
30 PRINT A+B
```

このように変更したプログラムを実行すると，まず次のような表示があらわれます。

```
RUN
A=? ■
```

このように，INPUT命令で，変数名の前に” ”で囲んだ文字列のことをプロンプト文と呼び，プロンプト文は実行時に？ の前に表示されます。これで，今なにを入力すれば良いのかがはっきりしますね。

INPUT ”プロンプト文”；変数名

プロンプト文と変数名の間には；（セミコロン）を書かなければなりませんので，気をつけてください。もちろん，プロンプト文はあなたが自由に決められます。
では，このプログラムを最後まで実行してみましょう。

```
RUN
A=? 1234
B=? 4321
 5555
Ok
```

PRINT命令を使いこなそう

さて，足し算したい2つの数をキーボードから入力すると，足し算をして答を表示するプログラムで，だいぶコンピューターらしくなってきました。ここでは，もうひと工夫して，もっと見やすい表示にする方法を考えてみましょう。加えて，プログラムの修正の方法についても練習していきます。

さて，今のプログラムでは，計算の答がいきなりパッと表示されておしまいです。そこで，これがA＋Bの答である，ということを知らせる表示を出すことにしましょう。そのために，プログラムを次のように修正したいと思います。

```
10 INPUT "A=";A
20 INPUT "B=";B
25 PRINT "A+B="
30 PRINT A+B
```

修正の内容は，行25の追加です。どうやって追加すれば良いかというと，「Ok」があらわれていることを確かめてから次のようにキーを押せば良いのです。

[2][5][　][P][R][I][N][T][　]["][A][＋][B][＝]["][RETURN]

これで，行25が追加されました。LIST命令で確認しましょう。

```
LIST
10 INPUT "A=";A
20 INPUT "B=";B
25 PRINT "A+B="
30 PRINT A+B
Ok
```

後から追加した行25が，ちゃんと行20と行30の間におさまっています。コンピューターは，プログラムの行をどんな順で入れてもきちんと行番号順に並べ換えてくれるのです。

また，このことによって，なぜ行番号を10，20，30のように10おきにつけたのかも分かりましたね。あとから行を間に追加できるようにしておくためだったわけです。

さて，この行25でなにをしているのかというと，行30で足し算の答を表示する前に，その答はA＋Bの答だ，ということがはっきり分かるように，「A＋B＝」という表示を出させているのです。論より証拠，このプログラムで100＋50の計算をして，どんな表示が出るか見てみましょう。

```
RUN
A=? 100
B=? 50
A+B=          ←行25の結果
 150
```

確かに，150の答の前に「A+B=」と表示されました。でも，どうせなら

```
A+B= 150
```

のように同じ行に表示された方が良いと思います。

セミコロン（；）で表示をつなげる

行25を，次のように修正してみましょう。

```
25 PRINT "A+B=";
                ↑
                └──これを追加
```

よく見ないと分かりませんが，最後に；（セミコロン）がつけ加えられています。
この修正を行うため，次の手順で操作してみてください。
まず，

```
LIST 25
```

と命令します。もちろん，最後に RETURN キーを押します。

```
LIST 25
25 PRINT "A+B="
Ok
```

LIST　行番号

LIST命令は，このように特定の行だけを表示させることもできるのです。
なお，参考までに

LIST　行番号—行番号　または　LIST　行番号—

という命令で，プログラムのある一部分だけを表示させることもできます。

さて，行25が表示されたら，カーソル（■マーク）をカーソルキーを使って行25の最後，つまりこれからセミコロン（；）を追加したい場所にもっていきます。

```
25 PRINT "A+B="■
OK
```
(カーソルをここへ)

次に，[;]キーを押します。

```
25 PRINT "A+B=";■
```

画面上ではセミコロン（；）が追加されましたが，これで安心してはいけません。ここでも，やはり[RETURN]キーを押さなければなりません。**最初にプログラムを入力したときも1行入れるごとに[RETURN]キーを押したように，修正を行った場合も同じです。[RETURN]キーを押すことによって，はじめて修正後の行がコンピューターの記憶に入れられるのです。**

では，行25が修正されたかどうか，LIST命令で確認し，続いてRUN命令でプログラムの実行です。；（セミコロン）により，表示がつながるでしょうか。

```
RUN
A=? 100
B=? 50
A+B= 150
OK
```

たしかに，「A+B=」の表示と，答の150が同じ行にくっついてあらわれました。

ちょっとクールなカンマ（,）

こんどは，行25を次のように修正してください。

```
25 PRINT "A+B=",
```

では，セミコロン（；）がカンマ（,）に代わったところでRUNしてみましょう。

```
RUN
A=? 100
B=? 50
A+B=          ␣150
```
(14字分)　(先頭から15字目)

こんどは，「A+B=」と150がぴったりくっつくのではなく，150が同じ行の先頭から15字目の位置に表示されました。

プログラムの一部を消す　DELETE

今，行25と行30は次のようになっています。

```
25 PRINT "A+B=",
30 PRINT A+B
```

この2行は，次のように1行にまとめることができます。

```
25 PRINT "A+B=",A+B
```

では，先ほど行25を修正（カンマに変更）したのと同じ要領で，上のように再修正してください。

すると，もう行30は不要です。この行をコンピューターの記憶から消しましょう。

1行だけを消すときは，次の2つの方法があります。ひとつは，

```
DELETE 30
```

と命令する方法です。DELETE命令は，プログラム中の指定した行を消してくれます。この命令は，

DELETE（デリート）　行番号－行番号

という形で使うのが普通で，指定した行から行までを消してくれます。

今回のように，1行だけ消したいときは，わざわざDELETE命令を使わなくても，ただ単に次のようにキーを押すだけで消すことができます。

[3][0][RETURN]

画面の表示を消す　CLS

ところで，今のプログラムでは，1回RUN命令で実行するたびに画面の行が下へ下へと進みますが，前回の実行結果も残ったままです。実行したら，まず画面上の表示がいったん消えると，もっと見やすくなるに違いありません。画面の掃除の専門家はCLS命令です。（シーエルエス，またはクリア・スクリーンと読みます。）

CLS（クリアスクリーン）

では，このCLS命令をプログラムにつけ加えてください。行番号は5にしましょう。

```
LIST
5 CLS
10 INPUT "A=";A
20 INPUT "B=";B
25 PRINT "A+B=",A+B
```

このプログラムだと，実行するたびに画面が掃除され，いつも同じ場所で計算が行われるので，より見やすくなりました。

ジャンプする GOTO

GOTO命令とループ

新しい命令をひとつ覚えましょう。

プログラムに次の行を追加してください。

```
30 GOTO 10
```

この命令のおかげで何が起こるか，ともかくプログラムを実行してみましょう。

ゴートウ GOTO 行番号

GOTOは，指定した行番号にすすみなさい（あるいはもどりなさい）という命令です。プログラムは，小さい行番号の行から順に実行されるのが基本です。GOTO命令は，その基本の流れを変える働きをする命令です。

```
GOTO 10
```

という命令の場合，プログラムの流れをもう一度行10に戻す働きをします。

一般に，GOTO命令は，指定した行に自由にプログラムの流れを変えるために使います。このプログラムで，なぜ計算が無限に続くようになったかというと，行30のGOTO 10によりプログラムの流れは行10に戻ります。そして行10，20，25と順に実行されると，また行30のGOTO 10により行10に逆戻り…こうして，このプログラムは

```
┌→10
│  ↓
│ 20
│  ↓
│ 25
│  ↓
└─30
```

という繰り返しを無限に続けるのです。**この無限の繰り返しを止めるには，CTRL＋STOPを押すしかありません。（CTRLキーとSTOPキーを同時に押します。）**
このように，プログラムの流れがあるところでぐるぐる回るような状態をループとよんでいます。そして，この例のように無限に続くループのことを無限ループといいます。
無限ループを作ることができたのは，GOTO命令のおかげでした。

さて，ここまでの練習でプログラムというものがどんなものか，感じをつかんでいただけたものと思います。この章はこのくらいにして，さらに次の章で新しい命令，新しいプログラムの技術を練習して行きましょう。

第2章
もっとコンピューターらしく

ループからの脱出

●条件判断——IF-THEN
●プログラムの終了——END
●行番号の整理——RENUM

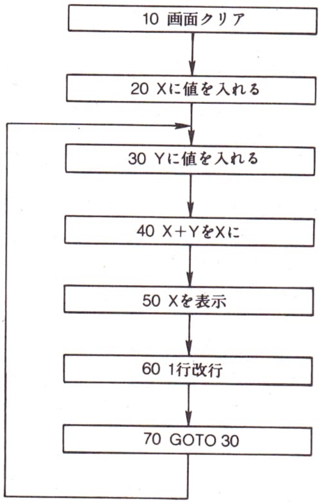
「もし〜なら」という条件判断　IF-THEN

まず無限ループから

無限ループのプログラムは，いったんRUN命令で実行してしまうと CTRL ＋ STOP キーで止めるまでいつまでも働き続けるのでした。ここでは，CTRL ＋ STOP キーを押さないでループから脱出する方法を考えます。CTRL ＋ STOP キーでループから脱出すると，プログラムの実行自体も終了してしまいます。それに対してここで考える方法ではループからは脱け出してもプログラムは続行，さらになにかの作業をすすめることもできるのです。

とりあえず，なにかひとつ無限ループのプログラムを作っておきましょう。

```
10 CLS
20 INPUT "START";X
30 INPUT "ANY NUMBER";Y
40 X=X+Y
50 PRINT "TOTAL=";X
60 PRINT
70 GOTO 30
```



(なお，プログラムの右に書かれた図をフローチャートと呼び，プログラムの流れを示したものです。順を追っていくとプログラムの流れがよく分かりますね。)

変数Yの値によってプログラム終了

わたしたちの日常生活で，ある条件によってその次の行動なり，できごとなりが決められることがたくさんあります。たとえば，晴れなら運動会決行，雨なら中止，とか，財布を調べてお金があれば映画を観に行く，なければ家で昼寝，夢でがまんする，など数えあげればきりがありません。ちょっとむずかしいことばを使うと，条件によって行動の判断をしていることになります。

ここではコンピューターにも条件判断をしてもらおうというわけです。

コンピューターに，ある条件を示してその条件によって次に行う命令を決めさせるのが，IF-THENという命令です。正しくは，IF-THENなどのようにコンピューターに対する指令を表すひとまとまりのことばをIF-THEN文，などのように文とよびます。PRINTやLET，INPUTも，みな文を作るためのことばなので，これからはPRINT文などのようによびます。

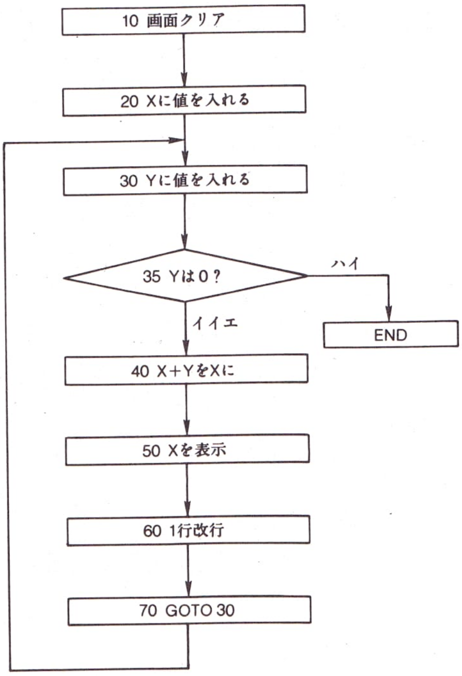
IF(イフ)　条件式　THEN(ゼン)　文

IF-THENは，条件式が正しければTHENの次の文を実行しなさい，という意味の文です。

では，[CTRL]＋[STOP]でプログラムをいったん止めて，次のIF-THEN文をプログラムに追加してください。

```
35 IF Y=0 THEN END
```

```
LIST
10 CLS
20 INPUT "START";X
30 INPUT "ANY NUMBER";Y
35 IF Y=0 THEN END
40 X=X+Y
50 PRINT "TOTAL=";X
60 PRINT
70 GOTO 30
```



追加した行35の意味は，「もしYが0なら，プログラムを終了しなさい」ということになります。

プログラムを終了させる　END

あらためて行35をみると，条件式はY＝0，文はENDです。

エンド
END

ENDは，プログラムを終了させる文です。
この文が実行されると，そこでプログラムは終了します。

条件式

一方の条件式ですが，条件式で使える記号をまとめると次のようになります。

記号	意　味	例	
＝	等しい	IF　A=B	AとBが等しければ
＞	大きい	IF　A>B	AがBより大きければ
＜	小さい	IF　A<B	AがBより小さければ
＞＝	大きいか等しい	IF　A>=B	AがBより大きいか等しければ
＝＞	大きいか等しい	IF　A=>B	AがBより大きいか等しければ
＜＝	小さいか等しい	IF　A<=B	AがBより小さいか等しければ
＝＜	小さいか等しい	IF　A=<B	AがBより小さいか等しければ
＜＞	等しくない	IF　A<>B	AとBが等しくなければ
＞＜	等しくない	IF　A><B	AとBが等しくなければ

行35の条件式の部分は

```
IF Y=0
```

ですから，「もしYと0が等しければ」つまり，「もしYが0なら」という意味になります。そして，この条件式が正しければ，THENの次に書かれた文が実行されます。この場合，ENDによりプログラムは終了します。

ひとこと

＝（イコール）記号は，LET文の場合右から左へ，という代入の方向を示していましたが，IF-THEN文の条件式の中で使われると「等しい」という意味になります。

行番号の整理　RENUM

さて，最初にプログラムを作ったときには行番号は10，20…70ときちんと10おきになっていましたが，あとから行35を追加したので，行番号がふぞろいになってしまいました。もちろん，これでもいいですし，行番号が1，3，28，29，78，105…のようにもっとふぞろいでもいっこうにかまわないのです。でも，やはり10おきにしたい，という場合や，たくさん行を追加したために，これ以上行が追加できなくなった場合，行番号をつけ直すための文があります。それがRENUM（リナム）文です。

では，直接モードで

```
RENUM
```

と命令し，次にLIST命令で行番号がどうなったか確かめてみましょう。

ちゃんと行番号が10おきに直されました。

繰り返しのプロフェッショナル

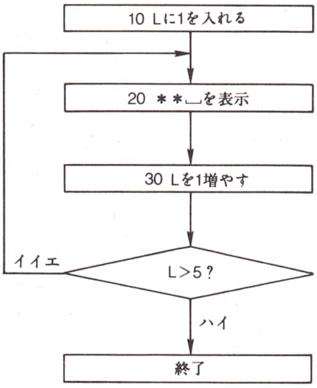
●ループを作る――FOR-NEXT

●ループの中にループ

回数を決めてループ　FOR-NEXT

前の項では，ループから脱出する手段としてIF-THEN文を覚えました。ループを繰り返す回数が決まっているときにはもっと便利な文があります。それを，実際のプログラムで覚えていきましょう。まず簡単なプログラムから始めます。

```
10 FOR L=1 TO 5
20 PRINT "** ";
30 NEXT L
```



このプログラムを実行すると，画面上に次のように表示されます。

```
RUN
** ** ** ** **
Ok
```

プログラムの行20のPRINT文は ＊＊␣ を1回表示させます。(␣はスペースを表わします。)

それに対して，プログラムの実行結果では＊＊␣が5組表示されています。ということは，行20が5回繰り返して実行されたことになります。その繰り返しを指令しているのが行10と行30の文です。

FOR（フォー）　変数＝始めの値　TO（トウ）　終わりの値 ｜ NEXT（ネクスト）　変数

FOR-NEXTは，**FORとNEXTの間にはさまれた部分を，指定回数繰り返しなさい**という意味の文です。

FORとNEXTはペアで

行10の

```
FOR L=1 TO 5
```

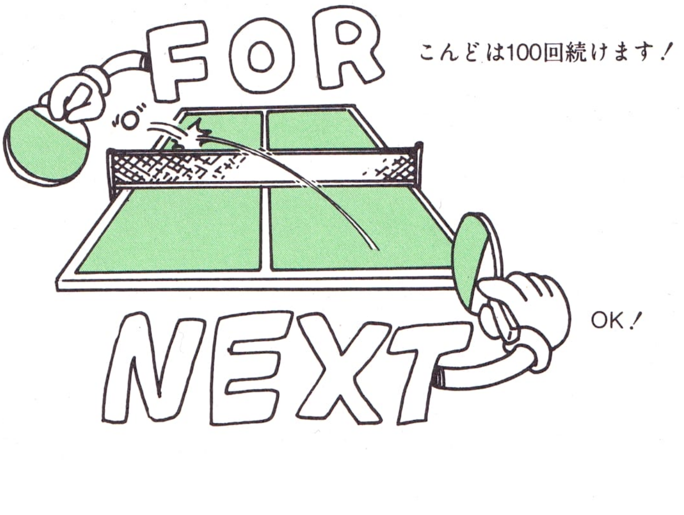
と，行30の

```
NEXT L
```

は，これで1組の対をなす文で，「Lの値を1から始めて1ずつ増やし，5になるまでこの2つの文にはさまれた部分のプログラムを繰り返しなさい。」という意味になります。つまり，このプログラムを実行すると，まずLには1が入ります。行10では，それ以外にLの最後の値は5であることもあらかじめ宣言しています。次に行20が1回実行され，行30にすすみます。ここで，Lの値を1増やし，Lの値が5になっていなければFOR文の次の行，つまり行20に戻ります。これが，Lの値が5になるまで繰り返されるのです。

このように，FOR文とNEXT文は，その間にはさまれた部分のプログラムを好きな回数だけ繰り返す働きをもっており，必ずペアで使われるので，まとめてFOR-NEXT文とよぶこともあります。また，FOR-NEXT文で作られるループをとくに**FOR-NEXT**ループなどということがあります。

なおFOR文とNEXT文中では，必ず同じ変数を使わなくてはなりません。（上の例ではL）



ループの中のループ

FOR-NEXT文はいろいろおもしろい使いかたができます。たとえば，ひとつのFOR文とNEXT文にはさまれたFOR-NEXTループの中に，別のFOR-NEXTループをおくことができます。

この場合，プログラムの構造はたとえば次のようになります。

```
FOR L=1 TO 10
 FOR M=1 TO 5
  FOR N=1 TO 7
  .
  .
  .
  .

  NEXT N
 NEXT M
NEXT L
```

ループ③　ループ②　ループ①

この例では，ループ①の内部にループ②があり，さらにその内部にループ③があります。ループ①は，10回繰り返されるようになっていますが，ループ①が1回繰り返されるごとにその内部のループ②は5回繰り返されます。そしてループ②が1回繰り返されるごとにその内部のループ③は7回実行されることになるのです。

プログラムの書きかたとしては，必ず1つのループごとに別の変数（上の例ではL，M，N）を使わなくてはなりません。

データがいっぱい

●プログラム内に変数の値を用意——READ-DATA

変数に値を入れるもうひとつの方法　READ-DATA

BASICプログラミングのコツのひとつは変数の扱いです。この変数にいろいろな値を入れて，計算や判断の材料として使ったりします。変数にある値を入れることを，値を代入する，といいます。

では，変数に値を代入するにはどんな方法があったでしょうか。それを思い出してみましょう。

まず，LET文による代入方法がありました。

変数Aに100という値を代入したければ

　LET　A＝100　（あるいは　A＝100）

というLET文を実行すれば良いですね。

次に覚えたのがINPUT文です。

INPUT文を使うと，LET文のようにプログラム中であらかじめ変数に代入する値を決めてしまうのではなく，INPUT文が実行された時点でキーボードを使って好きな値を代入することができました。

```
INPUT A
```

が実行され，？が表示されたときにキーボードから500を入力すれば変数Aに500が代入されます。

BASICでは，LET文，INPUT文のほかにもうひとつ，変数に値を代入するための文があります。それがREAD-DATA文です。

まず，ごく簡単なプログラムで実験です。

```
10 READ A
20 PRINT A
30 DATA 185
RUN
 185
OK
```

行10の

```
READ A
```

が実行されると，プログラム中のDATA文（この場合は行30にあります）に示された値（185）が変数Aに代入されます。（行20のPRINT文で，それを表示させています。）
文字型の変数の場合も同じです。

```
10 READ B$
20 PRINT B$
30 DATA SONY
RUN
SONY
Ok
```

SONYは文字型の値ですが，DATA文中では”　”で囲む必要はありません。
（ただし，文字列中にカンマを含んだり，文字列の先頭をスペースにしたい場合は文字列全体を”　”で囲まなければなりません。）

データの数をふやす

上の簡単なプログラムでは,1つの変数に1つの値を代入しただけです。こんどはデータ（値）を3つに増やしてみます。データを3つに増やすのだから，それを代入する変数も3種類用意しましょう。

```
10 READ A,B,C
20 PRINT A;B;C
30 DATA 10,20,30
RUN
 10  20  30
Ok
```

このように，ひとつのREAD文中に変数を，（カンマ）でつないで並べることができます。DATA文中にも，同じように代入したいデータを ， でつないで並べることができます。
大切なことは，READ文が要求する数だけは，DATA文によってデータが準備されていなければならないということです。

プログラムをディスクに記録するには

- ●ディスクのフォーマット——CALL FORMAT
- ●プログラムのセーブ——SAVE
- ●セーブを確認する——FILES
- ●プログラムのロード——LOAD
- ●ディスクからプログラムを消す——KILL
- ●プログラムをコピーする——COPY

ディスクのフォーマット　CALL FORMAT

せっかく作った大切なプログラムも，コンピューターの電源を切るとコンピューターの記憶から消えてしまいます。そこで，とっておきたいプログラムはコンピューターのディスクドライブ装置を使ってディスクに記録しておきましょう。そうすれば，コンピューターのスイッチを切っても，次に電源を入れたときにそのディスクからもう1度プログラムをコンピューターの記憶に読み込むことができます。

ところで，フロッピーディスクですが，買ったままの状態ではすぐにプログラムを記録する（書き込む，セーブするともいいます）ことはできません。プログラムをセーブする前に，ディスクをフォーマットしておく必要があります。

フォーマットという作業は，ディスク上に特定のデータをあるルールに従って書き込むことで，コンピューターはこのデータを目印にしてディスク上のいろいろな場所を探しあてるのです。いわば，白い紙に線を引いて文字を書く場所や間隔を決めるようなものです。ただし，みなさんはフォーマットによってどんなデータが書き込まれるのかを知る必要は特にありません。ただ，新しいディスクは必ずフォーマットしてから使う，ということを覚えておいてください。

それでは，新しいディスクを1枚用意し，次の手順を実行してください。

1　CALL FORMATという命令を実行します。

コール　フォーマット CALL FORMAT

最後に[RETURN]キーを押すと，

```
Drive name ? (A,B)
```

という表示があらわれます。これは，Aドライブのディスクをフォーマットしますか，Bドライブのディスクをフォーマットしますか，とたずねてきているのです。ディスクドライブが1台のときはこれはAドライブですから[A]キーを押します。

```
1 - Single sided, 9 sectors
2 - Double sided, 9 sectors
```

という表示が出ます。

2　フロッピーディスクが片面だけ使えるタイプのものの場合は[1]キーを押します。両面使えるタイプのもののときは[2]キーを押します。キーを押すと，次の表示が出ます。

```
Strike a key when ready
```

3　新しいディスクをディスクドライブに入れ，キーボード上のキーをひとつ押します。するとディスクのフォーマットが始まり，しばらくしてフォーマットが終わると

```
Format complete
Ok
```

という表示があらわれます。

プログラムのセーブ　SAVE

コンピューターが記憶しているプログラムをディスクに中間言語形式で記録（セーブ）するにはSAVE命令を使います。

SAVE "A:ファイル名.タイプ名"

A:は，Aドライブを指定するための記号ですが，ディスクドライブが1台の場合は省略してかまいません。
ファイル名というのは，みなさんがプログラムにつける名前のことで，この名前でいろいろなプログラムを区別します。

HEART……これは5字の名前

などとつけます。ただし，名前をつけるときに守らなくてはならないルールがあります。

ファイル名をつけるときのルール

- ファイル名は8文字までです。
- アルファベットの他，カタカナ，ひらがな，数字や記号も使えます。
- アルファベットの大文字と小文字は区別されません。
- ,.;:"*?およびスペースは使えません。

タイプ名というのは，**ファイル**（ディスクに記録されるひとまとまりの内容）の種類を示すもので，ファイル名のあとに.（ピリオド）をはさんで3文字以内で自由につけることができます。今記録しようとしているのはBASICのプログラムなので，BASというタイプ名が適当でしょう。このタイプ名は省略してもかまわないのですが，みなさんがディスクの扱いに慣れていろいろな種類のファイルをディスクに記録するようになったとき，つけておくと大変便利なものです。そこで今のうちからBASICのプログラムには，BASというタイプ名をつける習慣をつけておくと良いでしょう。

ここでは，セーブするプログラムのファイル名をHEART，タイプ名をBASとすることにして，このプログラムをディスクに記録（セーブ）する例をみていきましょう。
次のように命令します。

```
SAVE "HEART.BAS"
```

[RETURN]キーを押すとディスクドライブが作動し，プログラムがセーブされます。
セーブが終了するとOkが表示されます。

セーブされたか確認　FILES

プログラムがディスクにセーブされたかを確かめる命令があります。

ファイルズ FILES

FILESは，ディスクにセーブされたファイルのファイル名とタイプ名を表示させる命令です。

```
FILES
HEART    .BAS
Ok
```

今セーブしたプログラムのファイル名HEART.BASが表示されましたね。さらに，このディスクに別のプログラムを別のファイル名でセーブすれば，FILES命令によってそのプログラムのファイル名も表示されることになります。

プログラムのロード　LOAD

プログラムをディスクからコンピューターの記憶に読み込む（ロードする）前に，いったんNEW命令でコンピューターの記憶を消しておきましょう。
プログラムをディスクからロードするにはLOADという命令を使います。

ロード LOAD　“A:ファイル名.タイプ名”

A:は省略することができます。ファイル名，タイプ名は，セーブしたときと同じものを指定します。したがって，

```
LOAD "HEART.BAS"
```

と命令すれば良いことになります。RETURNキーを押すとディスクドライブが作動してプログラムのロードが始まります。ロードが終了するとOkがあらわれますからLIST命令で確認し，RUN命令で実行してみましょう。

これで，ディスク上のプログラムはいつでも好きなときにLOAD命令でコンピューターの記憶にロードして実行することができることが分かりました。

不要なプログラムを消す KILL

たくさんプログラムを作って，ディスクにセーブしていくとだんだんディスクがいっぱいになってきます。もういらないプログラムをディスクから消すには，KILL命令を使います。

KILL（キル） "A:ファイル名.タイプ名"

A:は省略することができます。今ディスクにセーブされているHEART. BASというプログラムをディスクから消すには

```
KILL "HEART.BAS"
```

と命令します。

プログラムをコピーする COPY

プログラムをあるディスクから別のディスクに，あるいは同じディスクにコピーするときは，COPY命令を使います。

COPY（コピー） "ドライブ名1：ファイル名.タイプ名" TO "ドライブ名2：ファイル名.タイプ名"

ドライブ名1：，ドライブ名2：は，実際にはA：，B：のように指定します。ドライブ名1からドライブ名2へコピーされます。ドライブ名について詳しくは206ページをご覧ください。

1ファイルだけコピーする場合

●同一ディスクへコピーするときは

```
COPY "ABC.BAS" TO "XYZ.BAS"
```

このように，ドライブ名1，ドライブ名2ともに省略すると，同一ディスク上に別のファイル名でコピーすることができます。この例では，"ABC.BAS"という名のファイルと同じ内容のファイルが，"XYZ.BAS" というファイル名で同じディスク上に作られます。

```
COPY "ABC.BAS"
```

のように，TO以下を省略すると，同一ファイル名で同一ディスク上にコピーをしようとしたことになり，エラーになります。

●別のディスクへコピーするときは

<u>ディスクドライブが1台の場合</u>

ディスクドライブが1台でも，これを仮想的にA：B：2台のドライブがあるものとみなして使うことができます。

```
COPY "A:ABC.BAS" TO "B:"
```

を実行すると，

```
Insert diskette for drive B:  }—ⓐ
and strike a key when ready
```

というメッセージがあらわれます。ここで，今ドライブに入っているディスクを抜き取り，新しいディスク(フォーマットされているもの)を入れてなにかひとつキーを押します。すると，

```
Insert diskette for drive A:  }—ⓑ
and strike a key when ready
```

というメッセージがあらわれます。新しいディスクを抜き取り，もとのディスクを入れて，なにかキーを押します。そして，ⓐのメッセージがあらわれたら新しいディスクを，ⓑのメッセージがあらわれたらもとのディスクを入れる，という作業をくり返します。OKがあらわれたら終了で，このとき新しいディスクに“ABC.BAS”というファイルが同じファイル名でコピーされています。

なお，くり返しの回数はコピーしたいファイルの長さによって異なります。

```
COPY "A:ABC.BAS" TO "B:XYZ.BAS"
```

のようにして同じ作業をすると，新しいディスク上にコピーされたファイルには，“XYZ.BAS”と名前が付け変えられます。

ディスクドライブが2台以上の場合

ドライブ名1，ドライブ名2で，どのディスクからどのディスクへコピーするのかを指定します。

```
COPY "A:ABC.BAS" TO "B:XYZ.BAS"
```

これにより，Aドライブのディスク中の“ABC.BAS”が，Bドライブのディスク上に“XYZ.BAS”のファイル名でコピーされます。

```
COPY "A:ABC.BAS" TO "B:"
```

これは，Aドライブのディスクの“ABC.BAS”を，Bドライブのディスク上に同一ファイル名（“ABC.BAS”）でコピーするための文です。

```
COPY "A:ABC.BAS" TO "XYZ.BAS"
```

のように，ドライブ名2を省略すると，現在使用中のディスクドライブのディスク上にコピーが行われます。

```
COPY "A:ABC.BAS"
```

のように，TO以下を省略すると，現在使用中のディスクドライブのディスク上に，同一ファイル名でコピーが行われます。したがって，上の例では現在使用中のディスクはAドライブ以外でなければなりません。現在使用中のディスクがAドライブだとエラーになります。

複数のファイルコピー

?（クエスチョンマーク）でフィイル名，タイプ名の1字分ならなんでもあらわすことができます。*（アステリスク）は，すべてのファイル名あるいはすべてのタイプ名の代用として使えます。このことを利用して，1回のCOPYコマンドで，複数のファイルをコピーすることができます。

●ディスクドライブが1台の場合

```
COPY "A:*.BAS" TO "B:"
```

により，あるディスク上にあるファイルのうち“BAS”というタイプ名のついたファイルをすべて別のディスク上にコピーすることができます。このとき，ひとつのファイルのコピーでもそうだったように，メッセージにしたがってもとのディスクと新しいディスクを入れ替える作業を，OKがあらわれるまで続けます。

```
COPY "A:ABC.*" TO "B:"
```

とすれば，あるディスクの上の，“ABC”というファイル名のファイルをすべて別のディスクにコピーできます。

```
COPY "A:PROG??.BAS" TO "B:"
```

とすれば，あるディスク上で，ファイル名がPROGとあと2文字で，タイプ名が“BAS”のもの(PROG01.BAS，PROG02.BAS，PROGAB.BASなど)をすべて別のディスク上にコピーできます。

```
COPY "A:*.*" TO "B:"
```

とすれば，あるディスク上のファイルをすべて別のディスク上にコピーできます。ディスクドライブが1台の場合は，いずれのときもコピーが終了してOKが出るまでメッセージにしたがってもとのディスクと新しいディスクを入れ替える作業をくり返す必要があります。

●ディスクドライブが2台以上の場合

COPYコマンドの使いかたは，ディスクドライブが1台のときと同じです。ドライブ名1でコピーするファイルの入ったドライブを，ドライブ名2でコピーをとりたい新しいディスクの入ったドライブを指定します。

```
COPY "A:*.*" TO "B:"
```

により，Aドライブのディスクのすべてのファイルが，Bドライブのディスクにコピーされます。

第3章
配列変数を使おう

配列変数を使ったプログラム

●配列変数の宣言——DIM

配列変数の使いかた　DIM

実際のプログラムで配列変数を使ってみましょう。

ここでは，N$（L）とT$（L）というふたつの文字型配列変数を使うことにします。どちらも，5つ分の変数になるようにします。ということは，Lの値は0から4までがとれることになります。

そして，これらの配列変数にどんなデータを入れるのかというと，N$(L)には人の名前，T$(L)には電話番号を入れることにします。入れるデータは，具体的には次のようにすることにしましょう。

配列変数	データ	配列変数	データ
N$(0)	PETER	T$(0)	111—2222
N$(1)	PAUL	T$(1)	222—3333
N$(2)	MARY	T$(2)	333—4444
N$(3)	TOM	T$(3)	444—5555
N$(4)	SUSIE	T$(4)	555—6666

データを配列変数に代入

配列変数にデータを代入するときもふつうの変数と同じようにLET文，INPUT文，READ-DATA文が使えます。ここでは，READ-DATA文を使ってみましょう。

```
10 DIM N$(4),T$(4)
20 FOR L=0 TO 4
30 READ N$(L),T$(L)
40 NEXT L
100 END
110 DATA PETER,111-2222
120 DATA PAUL,222-3333
130 DATA MARY,333-4444
140 DATA TOM,444-5555
150 DATA SUSIE,555-6666
```

プログラムの行10にいきなり新しい文が登場しました。

```
10 DIM N$(4),T$(4)
```

DIM文は，そのプログラムの中でどれだけの個数を持つ配列変数を使うかをあらかじめ決めておくために必要な宣言文です。DIM N$(4),T$(4)では，N$(0)からN$(4)までの5つの個数を持つN$という配列変数と，T$(0)からT$(4)までの5つの個数を持つT$という配列変数を使う，と宣言しているのです。

DIM（ディム） 配列変数名（最大値）

このように，DIM文では配列変数の（　）内の数値の最大値を指定します。指定した最大値をLとすると，0からLまでの，Lより1大きい数だけの配列変数が用意されることになります。

配列変数を使うときは，必ずDIM文で宣言してから使う，ということが重要ですので，忘れないでください。

さて，配列変数の用意ができたところで，行20～40のFOR—NEXTループで配列変数にデータを代入しています。第1回目のループではLの値は0ですから，行30のREAD文によりN$(0)にPETERという文字型のデータが，T$(0)に111—2222という，やはり文字型のデータが入ります。

そして，N$(1),T$(1),N$(2),T$(2)と次々にデータが代入され，ループの最後ではN$(4)にSUSIEが，T$(4)に555—6666が代入されてこのプログラムは終了します。

このプログラムを実際に実行してみたところで，本当にこれらの配列変数にデータが正しく代入されたか，直接モードでPRINT文を使って確認しましょう。
まず，

```
PRINT N$(0),T$(0)
```

としてみてください。

```
PRINT N$(0),T$(0)
PETER          111-2222
OK
```

たしかに，N$(0)にはPETERが，T$(0)には111—2222が入っていますね。
また,PRINT　N$(1)によってPAULが表示されますし,PRINT　T$(2)によって333-4444が表示されるはずです。

——————*——————*——————*——————*——————

さて，「入門編」はこれではおしまいです。
コンピューターとプログラムの関係が分かっていただけたでしょうか。また,BASICでプログラムを作って実行させる方法はよく分かりましたか。それさえ分かれば「入門編」の目的は充分達せられたのです。あとは，みなさんが今まで覚えた命令を組み合わせて自由にプログラムを，それもできるだけ多く作ってみてください。初めは,「入門編」の中に出てきたプログラムの一部を，ほんのちょっと手直しするだけでもかまいません。それによって思わぬ結果になることもあります。また，自分でプログラムを作るときは，何よりもまちがいを恐れないことです。プログラムがまちがっていてもコンピューターはこわれません。それどころか，どこにどんなまちがいがあるのかをエラーメッセージで教えてくれるのですから,もしエラーメッセージが出てきたら，LIST命令でプログラムを表示し，どこがまちがっているのかじっくり探しましょう。

応用編

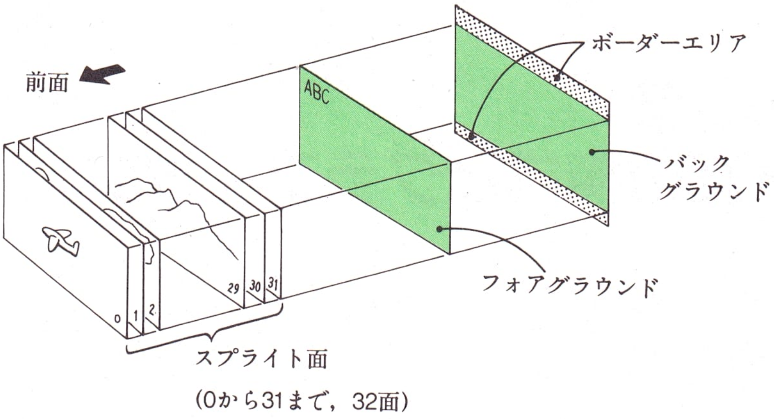
第4章
画面の構成とグラフィック

画面のモード

- 画面の構成
- モードの設定—SCREEN
- 1行の文字数の指定—WIDTH
- グラフィックモードの座標とSTEPの指定

画面の構成

MSX$_{2+}$-BASICでは，画面の構成は次のようになっています。



画面モードの設定　SCREEN

文字(アルファベットや数字など)を表示させるのがテキストモードで，図形をドット単位で表示させるのがグラフィックモードです。MSX$_{2+}$-BASICでは，2つのテキストモードと10のグラフィックモードの計12の画面モードがあり，SCREEN文によっていずれかのモードを選択します

SCREEN（スクリーン） [画面モード],[スプライトの大きさ],[キークリックスイッチ],[ボーレート],[プリンタータイプ],[インターレースモード]

SCREEN文は，画面のモード，スプライトの大きさ，キークリック音の有無，カセットテープレコーダーのボーレート，プリンターのタイプ，インターレースモードを設定します。
画面のモード以外については，107ページ以降で説明します。

スクリーン文によって選択できるモードをまとめると次のようになります。

モード番号	モード	画面表示	カラー	ページ	スプライト
0	テキスト	横最大80字，たて24行 かな表示不可	カラーパレット機能 16色/512色	—	使えない
1		横最大32字 たて24行 かな表示可	カラーパレット機能 16色/512色	—	使える
2	グラフィック	256×192ドット	カラーパレット機能 16色/512色 (2色/8ドット)	—	使える
3		256×192ドット マルチカラー	カラーパレット機能 16色/512色	—	使える
4		256×192ドット	カラーパレット機能 16色/512色 (2色/8ドット)	—	使える (強化スプライト)
5		256×212ドット	カラーパレット機能 16色/512色	4ページ	使える (強化スプライト)
6		512×212ドット	カラーパレット機能 4色/512色	4ページ	使える (強化スプライト
7		512×212ドット	カラーパレット機能 16色/512色	2ページ	使える (強化スプライト)
8		256×212ドット	256色	2ページ	使える (強化スプライト)

モード番号	モード	画面表示	カラー	ページ	スプライト
10	グラフィック	256×212ドット	表示可能色 12499色 指定可能色 カラーパレット機能 16色/512色	2ページ	使える（強化スプライト）
11		256×212ドット	表示可能色 12499色 指定可能色 コード数 256	2ページ	使える（強化スプライト）
12		256×212ドット	表示可能色 19268色 指定可能色 コード数 256	2ページ	使える（強化スプライト）

SCREEN 10, SCREEN 11およびSCREEN 12モードでのカラー表示については71ページをご覧ください。

どのモードでも，フォアグラウンドとバックグラウンド，ボーダーエリアを使います。このうち，文字や図形が表示されるのはフォアグラウンド面で，バックグラウンドとボーダーエリアはその色だけを変えることができます。
また，SCREEN 0のモード以外では，これに加えてスプライト面を使用することができます。スプライト面とは，自由に形を定義できるスプライトパターンを使って動画などを表示する面のことです。スプライトの使いかたについては，次の章で説明します。

キャラクターモード

キャラクターモードとは，文字を表示するモードです。
SCREEN 0とSCREEN 1で指定されるのがキャラクターモードです。
SCREEN 0のモードでは，1文字が6ドット幅×8ドットで表示され，SCREEN 1のモードでは，1文字が8ドット幅×8ドットで表示されます。
MSXコンピューターのかな文字やグラフィックキャラクターの一部分は，幅が8ドットあります。したがって，かなやグラフィックキャラクターを使うときはSCREEN 1のモードにしてください。

1行の文字数の指定　WIDTH

SCREEN 0のモードでは，1行最大80字まで，SCREEN 1のモードでは，1行最大32字まで表示できます。
1行の表示文字数は，WIDTH文で指定できます。

WIDTH（ウイドス）　表示文字数

SCREEN 0の場合，1行の表示文字数が1～40字のときと，41～80字のときでは文字の幅が異なります。41～80字のときの文字幅は1～40字のときの半分になります。

SCREEN 0：WIDTH 40を実行すると	SCREEN 0：WIDTH 80を実行すると
ABCDEFG	ABCDEFG

これらの2つを区別するため，1行の文字数が1～40字の状態を40字モード，41～80字の状態を80字モードと呼びます。
SCREEN 1の場合は，文字数1から32まで，文字幅は同じです。

```
                    ┌ SCREEN 0 ─┬ 80字モード ←┐ WIDTH文で切り換え
キャラクターモード ─┤            └ 40字モード ←┘
                    └ SCREEN 1  ── 32字モード
```

文字数と表示領域

SCREEN 0の80字モードでWIDTH 80を実行した場合，およびSCREEN 0の40字モードでWIDTH 40を実行した場合，それにSCREEN 1のモードでWIDTH 32を実行した場合，文字は画面の左右いっぱいに表示され，それぞれのモードで1行の文字数を減らしていくと文字の表示領域が中央部にせばまっていきます。

SCREEN 0：WIDTH 30では	SCREEN 1：WIDTH 10では
ABCDEFG	ABCDEFG

次のプログラムは，各モードでの1行の文字数をいろいろ変化させたときの画面ようすをみるためのものです。

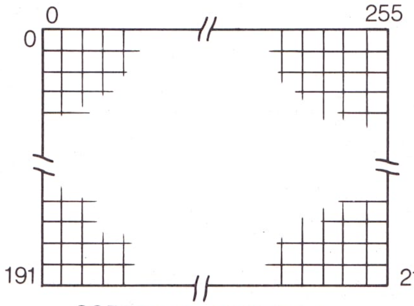
```
10 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 B$="abcdefghijklmnopqrstuvwxyz"
30 SCREEN 0:CLS
40 FOR W=80 TO 10 STEP -10
50 WIDTH W
60 GOSUB 150
70 NEXT W
80 SCREEN 1
90 FOR W=32 TO 12 STEP -10
100 WIDTH W
110 GOSUB 150
120 NEXT W
130 SCREEN 0:WIDTH 40
140 END
150 PRINT "WIDTH";W
160 PRINT:PRINT:PRINT
170 PRINT A$;B$
180 FOR T=0 TO 1000:NEXT T
190 RETURN
```

グラフィックモードと座標

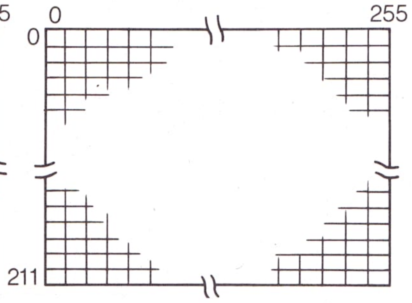
グラフィックモードは，SCREEN 2～SCREEN 8およびSCREEN 10～SCREEN 12で指定します。いずれのモードでも，次のような文により図形を描くことができます。

PSET, PRESET…点を打つ
LINE…直線または四角形を描く
CIRCLE…円，だ円，円孤，扇型を描く
PAINT…塗りつぶす
DRAW…任意の図形を描く

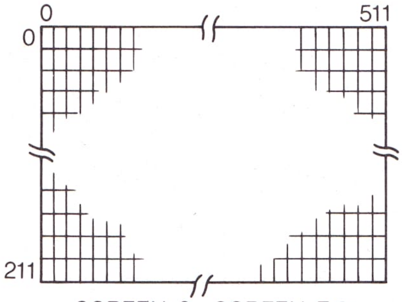
これらの文を使うとき，画面上の位置を指定するために画面には座標が設定されています。座標はモードによって異なります。



SCREEN 2，SCREEN 3，
SCREEN 4の場合



SCREEN 5，SCREEN 8，
SCREEN 10～12の
場合



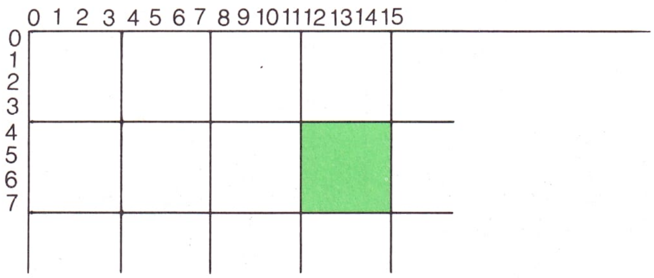
SCREEN 6，SCREEN 7の
場合

次のプログラムで，SCREEN 2，SCREEN 5，SCREEN 6の各モードで，同じ座標値を使った円が画面上のどの辺に描かれるか調べてみましょう。

```
10 SCREEN 2
20 GOSUB 100
30 SCREEN 5
40 GOSUB 100
50 SCREEN 6
60 GOSUB 100
70 END
100 CIRCLE (125,100),90
110 FOR T=0 TO 1000:NEXT T
120 RETURN
```

マルチカラーモード（SCREEN 3）

SCREEN 3のモードでは，SCREEN 2，SCREEN 4と同じ256×192ドットの座標系を使いますが，描かれる図形は4×4ドットのブロックが単位になります。



たとえば，

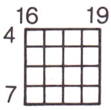
```
PSET (12,4),1
PSET (14,5),1
PSET (15,7),1
```

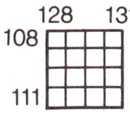
はすべて同じブロック内の1ドットを指定しているため，結果としてどれも図の■の部分を黒く塗りつぶすことになります。

また，

```
LINE (17,5)-(130,110)
```

は（17，5）と（130，110）を含むブロック，つまり



と


を結ぶ荒い線を引きます。

マルチカラーモードを使ったプログラムをひとつ実行してみましょう。

```
10 SCREEN 3
20 COLOR ,10,10:CLS
30 LINE (80,76)-(176,108),8,BF
40 FOR Y=80 TO 104 STEP 4
50 FOR X=84 TO 172 STEP 4
60 READ P
70 IF P=0 THEN SET BEEP 1 ELSE 100
80 PSET (X,Y),15:BEEP
90 GOTO 120
100 SET BEEP 2
110 PSET (X,Y),4:BEEP
120 NEXT X
130 NEXT Y
140 GOTO 140
150 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0
160 DATA 0,1,0,1,0,1,1,1,0,1,0,0,0,1,0,0
,0,1,1,1,0,1,0
170 DATA 0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,0
,0,1,0,1,0,1,0
180 DATA 0,1,1,1,0,1,1,0,0,1,0,0,0,1,0,0
,0,1,0,1,0,1,0
190 DATA 0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,0
,0,1,0,1,0,0,0
200 DATA 0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1
,0,1,1,1,0,1,0
210 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0
```

行10のSCREEN 3文でマルチカラーモードを指定しています。ループ内では，行60のREAD文で変数Pにデータを代入し，その値が0かそうでないかによって，PSET文で描く点（ブロック）の色を15（白）または4（濃い青）にしています。また，このときやはりPの値によって前の章に出てきたSET BEEP文で音を変え，その音を1回PSET文が実行されるごとに出して音の効果も利用しています。
次のような絵が描けましたか。

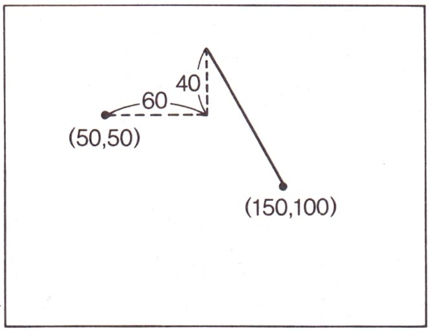
STEPの指定

PSET，PRESET，LINE，CIRCLE，PAINTおよび次章で説明するPUT SPRITE文では，(X，Y) により座標を指定するときに**STEP (X，Y)** という指定のしかたをすることができます。

これらのグラフィック関係の文を実行すると，最後に指定した点が覚えられます。そして次にSTEP (X，Y) と指定すると，最後に指定された点を原点 (0，0) とみなして，この新しい座標系上で (X，Y) の位置が決められます。ただし，STEPを省略すれば，いつでも画面左上隅を原点とする通常の座標系上で位置を指定することができます。

```
10 SCREEN 2
20 PSET (50,50)
30 LINE STEP (60,-40)-(150,100)
40 GOTO 40
```

このプログラムでは，行20でPSET文を実行したときに使われた座標 (50，50) が覚えられて次の行30に進みます。ここでLINE文の始点の指定にSTEP (60，−40) という形が使われているので，(50，50) を新たな原点としてそこからX方向に60，Y方向に−40の位置が始点になります。



なお，各グラフィック文のSTEPを含めた形は次のとおりです。

PSET STEP (X, Y)，カラー

PRESET STEP (X, Y)，カラー

LINE STEP (X, Y) -STEP (X, Y)，カラー，B / BF

CIRCLE STEP (X, Y)，半径，カラー，開始角，終了角，たて横比

グラフィック画面に文字を出すには

グラフィックモードの画面にも文字を表示することができます。そのためには，グラフィックモードの画面をファイルデバイスとみなして，ファイルを開き，表示したい文字をファイルに出力します。詳しくは226ページをご覧ください。

カラーの指定

- パレット機能について
- パレットの設定―COLOR
- SCREEN 8モードのカラー
- 色バケについて（SCREEN 2とSCREEN 4）
- SCREEN 10，11，12モードとカラー
- カラーを初期状態に戻す―COLOR

カラーコードとパレット機能

SCREEN 2のモードでは，色は全部で16色使え，それぞれの色にカラーコードがつけられています。カラーコードと色の対応は次のようになっています。

コード	色	コード	色	コード	色	コード	色
0	透明	4	濃い青	8	赤	12	濃い緑
1	黒	5	淡い青	9	淡い赤	13	マゼンタ
2	緑	6	濃い赤	10	濃い黄	14	灰
3	淡い緑	7	空色	11	淡い黄	15	白

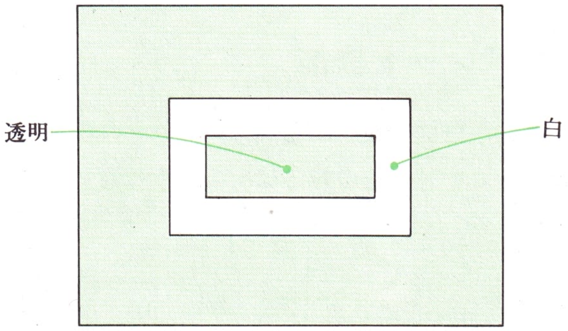
透明色について

この中で，コード0の透明という色について説明しておきます。透明とは，文字どおり透けてみえる，ということで，フォアグラウンドに透明を使って図形を描くと，その部分はバックグラウンドの色がみえることになります。

次のプログラムで確かめてみましょう。

```
10 SCREEN 2
20 FOR B=2 TO 14
30 COLOR ,B,B:CLS
40 LINE (50,50)-(180,140),15,BF
50 LINE (70,70)-(160,120),0,BF
60 FOR T=0 TO 1000:NEXT T
70 NEXT B
80 COLOR ,4,4:CLS
90 END
```

FOR-NEXTループにより，バックグラウンドおよびボーダーエリアの色は，コード2の緑からコード14の灰まで変化します。(行30のCOLOR文) それぞれのバックグラウンドの上に，白で大きい四角形を描き(行40)，次にその中に透明でひとまわり小さい四角形を描いています(行50)。すると，この透明の部分はいつもそのときのバックグラウンドの色と同じになります。すなわち，バックグラウンドが「透けてみえて」いるのです。



カラーパレット

カラーコード0から15までの16色が使えるのは，55ページの表からもわかるようにSCREEN 0，SCREEN 1，SCREN 2，SCREEN 3，SCREEN 4，SCREEN 5，SCREEN 7およびSCREEN 10の各モードです。

ところで，この項の最初に示した16色というのは，あくまでBASICがスタートしたときに使える16色ということであって，この16色が使える色のすべてではありません。

カラーコード0から15に対しては，512色の中から好きな色を選んで割り当てることができます。

512色といっても，そのひとつひとつに違った色の名前をつけることはできません。そこで，カラーコード1の色は赤が3，緑が2，青が7の割合で，というように，赤，緑，青の色の成分の配合で指定します。ちょうど，赤，緑，青の絵の具をパレットの上で混ぜ合わせるような感覚なので，これをパレット機能と呼んでいます。

ご注意

カラーコード0に対してもカラーを設定できますが，ちょっと特殊なので，ここではカラーコード1～15について説明します。

パレット機能の使いかた

赤，緑，青の各成分は，それぞれ0から7までの明るさの段階を持っています。これを輝度と呼びます。

各輝度は0から7の8段階指定できます。したがって，3つの輝度の組み合わせは8×8×8＝512とおりになります。

赤，緑，青の輝度をすべて0にすると黒になり，すべて7にすると白になります。また，赤，緑，青の輝度がすべて同じ（たとえば，すべて4）なら灰色になります。

赤	緑	青	色
0	0	0	黒
1	1	1	↑ 暗い灰
2	2	2	
3	3	3	
4	4	4	
5	5	5	
6	6	6	明るい灰 ↓
7	7	7	白

赤，緑，青の輝度のどれかひとつを他より大きくすると，その色相があらわれてきます。他の成分との差が大きければ大きいほど，また他の成分の輝度が小さいほど，純色に近づきます。いくつか例を挙げましょう。

赤	緑	青	色
4	3	3	赤味がかった灰色
5	2	2	赤に近い色
5	0	0	赤（やや暗い）
7	0	0	赤（純色，最も明るい）
2	0	0	黒に近い赤

パレットの設定　COLOR

カラーコードに，赤，緑，青の成分の輝度を設定して色を決めるのが，COLOR文です。

COLOR＝（カラーコード，赤の輝度，緑の輝度，青の輝度）

COLOR文では，赤，緑，青の成分の輝度を0〜7の数値で指定し，その結果得られる色を指定したカラーコードに対応する色とします。

たとえば，赤の輝度が4，緑の輝度が3，青の輝度が1の色をカラーコード5の色とするには

```
COLOR=(5,4,3,1)
```

を実行します。

```
10 SCREEN 5
20 COLOR=(1,7,7,7)
30 FOR C=2 TO 9
40 COLOR=(C,0,0,C-2)
50 NEXT C
60 COLOR ,1,1:CLS
70 FOR CC=2 TO 9
80 R=100-CC*10
90 CIRCLE (125,100),R,CC
100 PAINT (125,100),CC
110 NEXT CC
120 GOTO 120
```

このプログラムの行20から行50までで，カラーコード1から9までの色を設定しています。

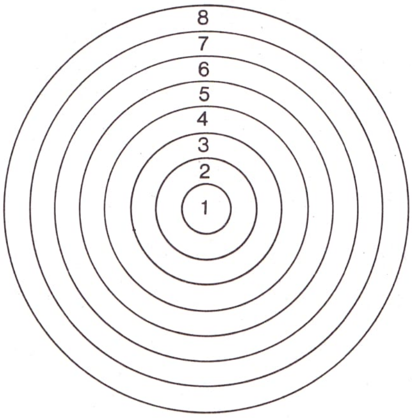
コード	赤	緑	青
1	7	7	7
2	0	0	0
3	0	0	1
4	0	0	2
5	0	0	3
6	0	0	4
7	0	0	5
8	0	0	6
9	0	0	7

コード1は白，コード2は黒，コード3から9までは黒に近い青から純色の青まで段階的に変化していきます。
したがって，行60のCOLOR文では，バックグラウンド，ボーダーエリアの色のコードを1にしていますが，これはもう黒ではなくて白になります。行70以下で，中心になるにしたがって純色の青になっていく円が描かれます。
このように，パレット機能を使うと，全体を同系色だけでそろえ，微妙な階調の変化をあらわすような絵も描けるようになります。

なお，このプログラムでは，SCREEN 5のモードを使用しています。SCREEN 2，4のモードはグラフィック文を実行したときの描くスピードが遅く，また色バケという問題が起こりうるからです。色バケについては後で説明します。
ではもうひとつ，パレット機能を使ったプログラムをみてみましょう。

```
10 SCREEN 5
20 FOR L=8 TO 1 STEP -1
30 CIRCLE (120,100),L*10+5,L
40 PAINT (120,100),L
50 NEXT L
60 K=(K+1) MOD 8
70 FOR L=1 TO 8
80 COLOR=(L,K,K,0)
90 K=(K+1) MOD 8
100 NEXT L
110 GOTO 60
```

行20～50のFOR-NEXTループで描かれる同心円のカラーコードは次のようになります。



最初，カラーコード1～8に対応する色は，黒，緑，淡い緑，濃い青，淡い青，濃い赤，空色，赤ですから，円は，これらの色で描かれていきます。
次に，パレット機能を使ってカラーコードに対応する色を次々に変えているのが行60～110の部分です。

行80のCOLOR文で，赤の輝度と緑の輝度を指定するのにKという変数が使われています。このKの値を決めているのが行60と90ですが，このなかで使われているMODとは，＋，－，＊，／などの演算符号のひとつです。
a＋bはaとbの足し算をしますが，**a MOD bはaをbで割った余りを求めます。**
したがって，Kと（K＋1），（K＋1）MOD 8の関係は次のようになります。

K	K＋1	（K＋1）MOD 8
0	1	1（1÷8＝0…1）
1	2	2（2÷8＝0…2）
2	3	3（3÷8＝0…3）
3	4	4（4÷8＝0…4）
4	5	5（5÷8＝0…5）
5	6	6（6÷8＝0…6）
6	7	7（7÷8＝0…7）
7	8	0（8÷8＝1…0）

変数は，特に値を代入しないと，その値は0になっています。したがって最初に行60が実行されると，その直前の変数Kの値は0ですから，行60によりKの値は1になります。そして，行90および行60で1回　K＝（K＋1）MOD 8が実行されるたびにKの値は1ずつ増え，7になると次は0に戻り，また7まで増えていきます。
そして，このKの値の変化と，行80のCOLOR文をはさむFOR-NEXTループにより，カラーコード1～8の色が，青の輝度だけ常に0で赤，緑の輝度が0～7の間で次々に変化していきます。
なお，行80を，たとえば

```
80 COLOR=(L,0,K,K)
```

に変更すれば，赤の輝度は常に0で，緑と青の輝度が変化することになります。同じように，緑の輝度だけ0にしたり，他にもいろいろな変更が可能ですので，試してみてください。

SCREEN 6モードとパレット機能

SCREEN 6のモードでもパレット機能は使えますが，カラーコードとして使えるのは0から3まで，つまり512色中4色が使えます。
なお，BASICスタート時のSCREEN 6モードでのカラーは次のように設定されています。

コード	色
0	透　明
1	黒
2	緑
3	明るい緑

SCREEN 8モードとカラー

SCREEN 8のモードにはカラーパレット機能はありません。そのかわり，256色が同時に使えます。
SCREEN 8モードでは，赤と緑の輝度が0から7までの8段階，青の輝度が0から3までの4段階，すなわち8×8×4＝256色あり，カラーコードは0から255までを使います。
カラーコードは，色の各成分の輝度から次の式で求められます。

カラーコード＝32×（緑の輝度）＋4×（赤の輝度）＋（青の輝度）

たとえば，緑の輝度が1，赤の輝度が5，青の輝度が3の色のカラーコードは

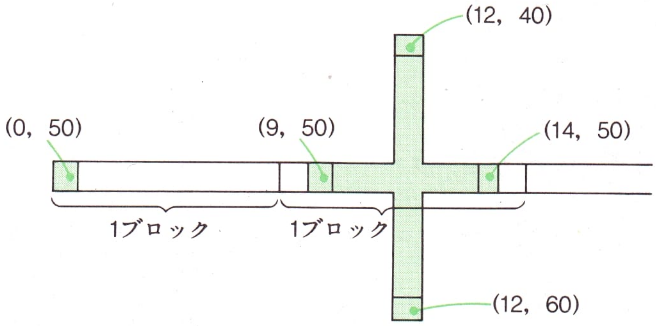
32×1＋4×5＋3＝55

で，55になります。

SCREEN 2とSCREEN 4モードの色バケについて

SCREEN 2とSCREEN 4のモードでは，横8ドットごとの範囲内だけに限って色を指定すると，2色しか指定できず，3色目を使うと2色目が3色目に変わる，という制限があります。

```
10 SCREEN 2
20 LINE (9,50)-(14,50),15
30 LINE (12,40)-(12,60),1
40 GOTO 40
```



このプログラムでは，行20で引いた直線が，横の座標が8から15までのブロック内に入っているので，色を白に指定したにもかかわらず，その次にこの線に交叉するようにして引いた黒の線のために，黒になってしまいます。

このように，指定した色と異なる色になってしまうことを色バケといい，SCREEN 2，SCREEN 4のモードを使うときは注意が必要です。

行20を

```
LINE (8,50)-(15,50)
```

にして，横の線がブロック内の8ドットいっぱいを使うようにすれば，白の指定が有効になります。

ただし，この次に

```
PSET (8,50),8
```

のように，さらに別の色でこの線上の色を変えると，ブロック内の8ドットすべてが最後に使った色（この場合，カラーコード8）に変わってしまいます。

すなわち，SCREEN 2およびSCREEN 4の1ブロック内（8ドット）では，最大2色しか使えません。

これに対して，SCREEN 5～8のモードでは1ドットに対して自由に1色を指定することができます。

SCREEN 10, SCREEN 11, SCREEN 12モードとカラー

YJK方式

SCREEN10～12のモードでは，カラーを表示するのにYJK方式と呼ばれる方式を使っています。YJK方式には，アトリビュート混在のYJK方式とアトリビュートなしのYJK方式があり，それぞれのモードで使用できるカラー数とYJK方式の種類をまとめると次のようになります。

モード	カラー	YJK方式
SCREEN 10	カラーパレット使用，512色中の16色＊	アトリビュート混在
SCREEN 11	0から255のコードで指定	アトリビュート混在
SCREEN 12	0から255のコードで指定	アトリビュートなし

＊アトリビュートビットが1にセットされていない画素（ドット）に対してロジカルオペレーション（PSET以外）を指定した場合は，カラーコードはパレット番号ではなくY成分を指定することになります。

使用できる色数は，SCREEN 10モードはSCREEN 5などと同じ，SCREEN 11とSCREEN 12はSCREEN 8と同じということになります。ただし，VRAMに書き込まれるデータからどのようなカラーを作り出すかという点で，YJK方式はSCREEN 8までの方式と違いがあります。したがって，実際に描画を行ったときにそれぞれで結果に違いが出てくる場合があります。

VRAMのデータとYJK方式

YJK方式では，画面を横256ドット，たて212ドットに分割します。そして各ドットに対して1バイト，つまり8ビットのデータを使ってその色を表わします。このとき，つねに横に連続した4ドットをひと組として色を表わすというルールがあります。

VRAM中の連続する4バイトを，アドレスの小さい順に0，1，2，3とすると，そのデータの内容は次のように表わされます。

ビット / アドレス	7	6	5	4	3	2	1	0
0	Y0					K_L		
1	Y1					K_H		
2	Y2					J_L		
3	Y3					J_H		

K_LはKの下位3ビット，K_Hは上位3ビットです。Jについても同様です。

Y0からY3まではそれぞれ5ビットですから0から31までの32とおりのデータを表わせます。KおよびJはそれぞれ6ビットで，符号つきで処理することにして−32から31までの64とおりのデータが表わせます。

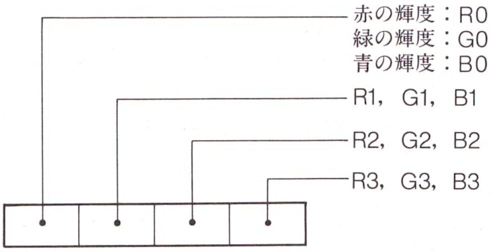
ここで，画面上に表示される4ドットのうち，1番左のドットのR（赤），G（緑），B（青）の輝度をR0，G0，B0とします。2番目から4番目のドットについても同様に，それぞれのR，G，Bの輝度をR1，G1，B1，R2，G2，B2，R3，G3，B3とします。すると，VRAM上のデータのY，J，Kの成分から，次の式で各輝度が求められます。

1ドット目のR，G，B

$R0=Y0+J$

$G0=Y0+K$

$B0=\frac{5}{4}Y0-\frac{1}{2}J-\frac{1}{4}K$



2ドット目のR，G，B

$R1=Y1+J$

$G1=Y1+K$

$B1=\frac{5}{4}Y1-\frac{1}{2}J-\frac{1}{4}K$

3ドット目のR，G，B

$R2=Y2+J$

$G2=Y2+K$

$B2=\frac{5}{4}Y2-\frac{1}{2}J-\frac{1}{4}K$

4ドット目のR，G，B

$R3=Y3+J$

$G3=Y3+K$

$B3=\frac{5}{4}Y3-\frac{1}{2}J-\frac{1}{4}K$

このようにして色を表わす方式を，アトリビュートなしのYJK方式といいます。

これに対して，アトリビュート混在のYJK方式では，VRAMのデータの各バイトのビット3をアトリビュートのビットとし，Yをビット7から4までの4ビットで表わします。

ビット / アドレス	7	6	5	4	3	2	1	0
0	Y0				A0	K_L		
1	Y1				A1	K_H		
2	Y2				A2	J_L		
3	Y3				A3	J_H		

A0からA3がアトリビュートビットです。そして，アトリビュートビットが0のときはそのバイトに対応するドットの色はアトリビュートなしのYJK方式と同様の計算式で求めることになります。また，アトリビュートビットが1のときはJとKの値には関係なく，Y0からY3までを各バイトに対応するドットの色のパレット番号（0～15）とみなします。したがって，アトリビュートビットが0のとき，YJK方式におけるYの値は0から30までの偶数ということになります。

なお，YJK方式を使ってMSX$_{2+}$-BASICで表現できる色の数は，計算途中の桁あふれやハードウェアの能力などの理由により，SCREEN 10，11で12499色，SCREEN 12で19268色となります。

SCREEN 10モードとSCREEN 5モードの違い

SCREEN 10モードで描画のコマンドを実行すると，ロジカルオペレーションをともなわないか，またはPSET（ロジカルオペレーションのPSET）であれば，結果はSCREEN 5モードと同様になります。ただし，VRAMのメモリー上ではふたつのモードで違いがあります。（ロジカルオペレーションについては，101ページに詳しく説明してあります。）

画面上の連続する8ドットに対応するVRAMの内容が，最初に次のようであったとします。（SCREEN 5のモードではバイト0から3まで，SCREEN 10のモードではバイト0から7までが画面上の8ドットに対応します。）

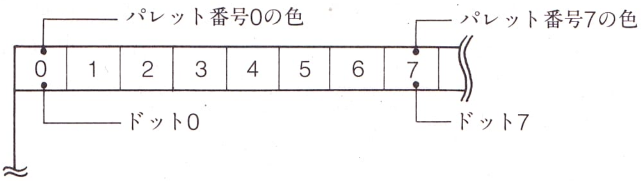
バイト	
0	0 1 0 1 0 1 0 1
1	1 0 1 0 1 0 1 0
2	0 1 0 1 0 1 0 1
3	1 0 1 0 1 0 1 0
4	0 1 0 1 0 1 0 1
5	1 0 1 0 1 0 1 0
6	0 1 0 1 0 1 0 1
7	1 0 1 0 1 0 1 0

このときに，次のプログラムを実行してドット0から7までにPSET文で色をつけることにします。（ドット0は画面座標上（0, 0）のドットです。）

```
10 SCREEN 5
20 FOR I=0 TO 7
30 PSET (I,0),I
40 NEXT I
```

このプログラム実行後のVRAMの内容は次のようになります。

バイト	上位	下位	
0	0 0 0 0	0 0 0 1	＝01（16進）
1	0 0 1 0	0 0 1 1	＝23
2	0 1 0 0	0 1 0 1	＝45
3	0 1 1 0	0 1 1 1	＝67
4	0 1 0 1	0 1 0 1	
5	1 0 1 0	1 0 1 0	
6	0 1 0 1	0 1 0 1	
7	1 0 1 0	1 0 1 0	



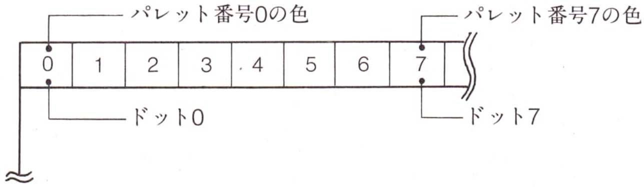
このように，バイト0の上位4ビットがドット0のパレット番号，下位4ビットがドット1のパレット番号，というように，バイト3の下位4ビットまででドット7までのパレット番号を表わします。したがって，バイト4以降のデータは変わりません。

これに対して，上のプログラムの行10を

```
10 SCREEN 10
```

として実行すると，VRAMの内容は次のように変わります。

バイト	上位4ビット	アトリビュートビット	下位3ビット
0	0 0 0 0	1	1 0 1
1	0 0 0 1	1	0 1 0
2	0 0 1 0	1	1 0 1
3	0 0 1 1	1	0 1 0
4	0 1 0 0	1	1 0 1
5	0 1 0 1	1	0 1 0
6	0 1 1 0	1	1 0 1
7	0 1 1 1	1	0 1 0



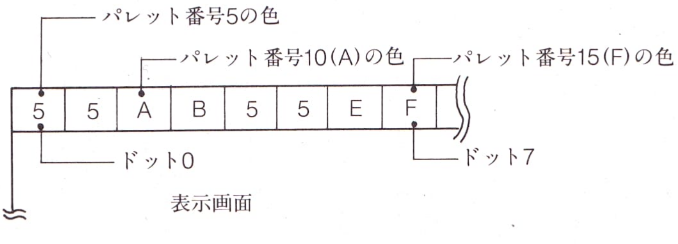
このように,バイト0から7までのそれぞれの上位4ビットがドット0から7までのパレット番号を表わすことになります。SCREEN 10はアトリビュート混在のモードであり,ロジカルオペレーションをともなわないか,PSETであれば,自動的にアトリビュートビットを1にセットします。したがって各バイトの上位4ビットはパレット番号を表わすことになるわけです。なお,各バイトの下位3ビットはプログラム実行前と変わりません。
これで同じ画面表示をするのに,SCREEN 10では,SCREEN 5の2倍のVRAMのメモリーが必要だということがお分かりいただけたと思います。

次に,PSET以外のロジカルオペレーションをともなう描画のコマンドを実行した場合です。ここでは,ロジカルオペレーションの例としてORを使ってみましょう。画面上の8ドットに対応するVRAMが,最初は74ページに示す状態だったとします。

```
10 SCREEN 5
20 FOR I=0 TO 7
30 PSET (I,0),I,OR
40 NEXT I
```

ここでこのプログラムを実行すると，VRAMのバイト0から3までの上位4ビット，下位4ビットの値が，順に0から7までの値（カラーコード）とORで演算されることになります。したがって，VRAMの内容は次のようになります。

バイト	上位	下位	
0	0 1 0 1	0 1 0 1	＝55（16進）
1	1 0 1 0	1 0 1 1	＝AB
2	0 1 0 1	0 1 0 1	＝55
3	1 1 1 0	1 1 1 1	＝EF
4	0 1 0 1	0 1 0 1	
5	1 0 1 0	1 0 1 0	
6	0 1 0 1	0 1 0 1	
7	1 0 1 0	1 0 1 0	



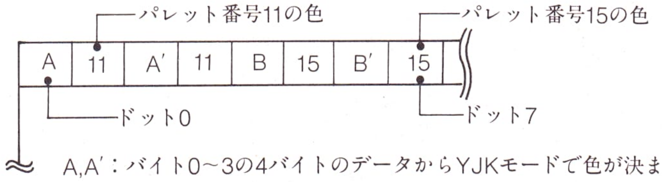
プログラム実行後のドット0から7の色は，それぞれパレット5，5，10(16進A)，11(B)，5，5，14（E)，15（F）の色ということになります。

また，プログラムの行10を

```
10 SCREEN 10
```

で実行した場合は，PSET文で指定されるカラーコード（0から7までを4ビットで表現した値）と，プログラム実行前のVRAMの各バイトの上位4ビットとの間でORの演算が行われます。アトリビュートビットは自動的に1にセットされませんのでビット3，および下位3ビットは変わりません。

バイト				
0	0 1 0 1	0	1 0 1	…ドット0はYJK方式で求められる色
1	1 0 1 1	1	0 1 0	…ドット1はパレット番号11(B)の色
2	0 1 1 1	0	1 0 1	…ドット2はYJK方式で求められる色
3	1 0 1 1	1	0 1 0	…ドット3はパレット番号11(B)の色
4	0 1 0 1	0	1 0 1	…ドット4はYJK方式で求められる色
5	1 1 1 1	1	0 1 0	…ドット5はパレット番号15(F)の色
6	0 1 1 1	0	1 0 1	…ドット6はYJK方式で求められる色
7	1 1 1 1	1	0 1 0	…ドット7はパレット番号15(F)の色



A,A′：バイト0〜3の4バイトのデータからYJKモードで色が決まる。
B,B′：バイト4〜7の4バイトのデータからYJKモードで色が決まる。

実行後のVRAMのアトリビュートビットに注目します。アトリビュートビットが0のバイトは0，2，4，6です。その結果，これらのドットはYJK方式で色が決まることになります。このとき，ドット0とドット2はバイト0から3までの値で，ドット4とドット6はバイト4から7までの値で色が求められます。

SCREEN 11，SCREEN 12モードとSCREEN 8モードの違い

ここでも，例としてVRAMが次のような状態であったときのことを考えてみます。SCREEN 8, 11, 12のいずれのモードでも，1バイトが画面上の1ドットに対応します。バイト0から7が，画面上のドット0から7に対応し，ドット0は（0，0）の位置にあるとします。

バイト	
0	0 1 0 1 0 1 0 1
1	1 0 1 0 1 0 1 0
2	0 1 0 1 0 1 0 1
3	1 0 1 0 1 0 1 0
4	0 1 0 1 0 1 0 1
5	1 0 1 0 1 0 1 0
6	0 1 0 1 0 1 0 1
7	1 0 1 0 1 0 1 0

まず，SCREEN 8モードでのカラー表示を調べるため，次のプログラムを実行した場合をみてみましょう。

```
10 SCREEN 8
20 PSET (0,0),240
30 PSET (1,0),248
```

ドット0に対して指定したカラーコードは240，ドット1のカラーコードは248ですから，プログラム実行後のVRAMは次のようになります。

バイト				G	R	B	
0	1 1 1	1 0 0	0 0	7	4	0	…ドット0
1	1 1 1	1 1 0	0 0	7	6	0	…ドット1
2	0 1 0	1 0 1	0 1	2	5	1	…ドット2
3	1 0 1	0 1 0	1 0	5	2	2	…ドット3
4	0 1 0	1 0 1	0 1	2	5	1	…ドット4
5	1 0 1	0 1 0	1 0	5	2	2	…ドット5
6	0 1 0	1 0 1	0 1	2	5	1	…ドット6
7	1 0 1	0 1 0	1 0	5	2	2	…ドット7

SCREEN 8モードでは，各バイトの上位3ビットで緑(G)の，次の3ビットで赤(R)の，下位2ビットで青(B)の輝度を表わします。したがって，画面上のドット0から7までの色のR，G，Bの各輝度は79ページに示すとおりになります。

次に，SCREEN 11モードの場合を調べるため，プログラムの行10を

```
10 SCREEN 11
```

として実行した場合を考えてみます。
プログラム実行後のVRAM内容は，SCREEN 8モードの場合と同じです。
なお，SCREEN11においては，ロジカルオペレーションの有無にかかわらず，アトリビュートビットが自動的にセットされることはありません。

バイト		アトリビュートビット	
0	1 1 1 1	0	0 0 0
1	1 1 1 1	1	0 0 0
2	0 1 0 1	0	1 0 1
3	1 0 1 0	1	0 1 0
4	0 1 0 1	0	1 0 1
5	1 0 1 0	1	0 1 0
6	0 1 0 1	0	1 0 1
7	1 0 1 0	1	0 1 0

ドット2：アトリビュートビットが0なので，YJK方式で色が決まる

ただし，SCREEN 11モードではアトリビュート混在のYJK方式で色が決まります。そこで，各バイトのアトリビュートビットに注目すると，ドット0，2，4，6はYJK方式で色が求められ，ドット1, 3, 5, 7は対応するバイトの上位4ビットの値をパレット番号とする色になることが分かります。その結果，バイト2のデータそのものは変わりませんでしたが，バイト0と1のデータが書き換えられたため，ドット2の色も変わってしまうことになります。(SCREEN 11モードでの色バケ) ドット3は，最初からアトリビュートビットが1のため，他のバイトの影響を受けず，色バケを起こしません。

こんどは，SCREEN 11モードでロジカルオペレーションを指定した場合をみてみます。この場合，VRAMのもとのデータと，描画のコマンドで指定した8ビットのカラーコードとの間で，8ビットどうしの演算が行われます。

```
10 SCREEN 11
20 PSET (0,0),240,OR
30 PSET (1,0),248,OR
```

このプログラムの実行の結果，VRAMのバイト0と1のデータが次のように変化します。

バイト		アトリビュートビット	
0	1 1 1 1	0	1 0 1
1	1 1 1 1	1	0 1 0
2	0 1 0 1	0	1 0 1
3	1 0 1 0	1	0 1 0

したがって，この場合もドット2（YJK方式で色が決まる）が，バイト0と1のデータの書き換えにともなって色バケを起こします。

最後に，SCREEN 12モードの場合を考えてみましょう。

```
10 SCREEN 12
20 PSET (0,0),240
30 PSET (1,0),248
```

このプログラム実行後のVRAM内容も，SCREEN 8，SCREEN 11モードの場合と同じです。

バイト

0	1 1 1 1 0	0 0 0
1	1 1 1 1 1	0 0 0
2	0 1 0 1 0	1 0 1
3	1 0 1 0 1	0 1 0
4	0 1 0 1 0	1 0 1
5	1 0 1 0 1	0 1 0
6	0 1 0 1 0	1 0 1
7	1 0 1 0 1	0 1 0

ただし，SCREEN 12モードではアトリビュートなしのYJK方式で色が決まります。したがって，バイト0とバイト1のデータが書き換えられたことにより，ドット2とドット3の色が変わってしまうことになります。(SCREEN 12モードでの色バケ)

初期状態に戻す　COLOR

COLOR文で，カラーコードとその色の設定を変更することができましたが，それをプログラム内でBASICがスタートしたときの状態に戻すには，

```
COLOR=NEW
```

という文を実行します。

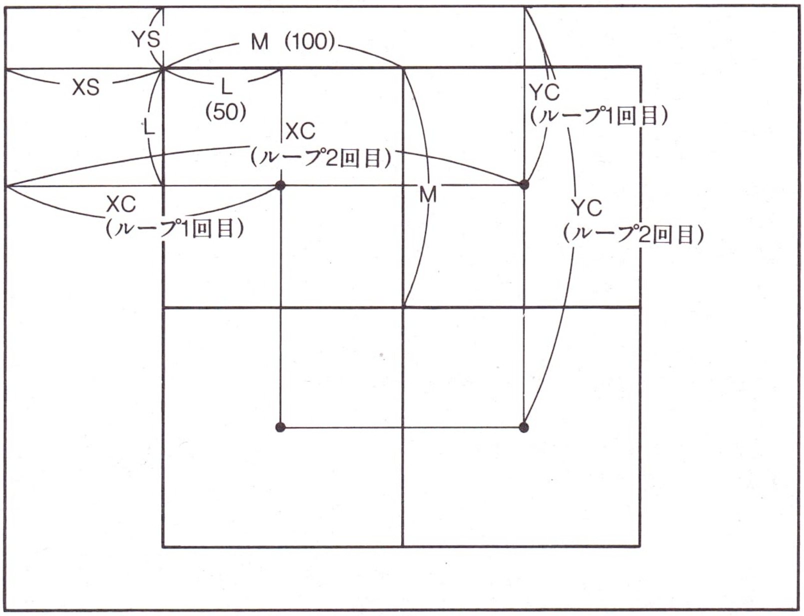
サンプルプログラム

次に，SCREEN 5モードでカラーパレット機能を使ったサンプルプログラムを示します。

```
10 S=2:CN=10:L=50:M=L*2
20 XS=(255 MOD M)/2:YS=(211 MOD M)/2
30 COLOR 15,0,0:SCREEN 5
40 '-
50 FOR T=0 TO TIME-INT(TIME/100)*100:J=R
ND(1):NEXT
60 ' *** draw box ***
70 FOR XC=L+XS TO 255-L STEP M
80  FOR YC=L+YS TO 211-L STEP M
90  C=0
100   FOR P=L TO 0 STEP -S
110    LINE(XC-P,YC-P)-STEP(P*2,P*2),C+1
,BF
120    C=(C+1)MOD CN
130   NEXT P
140  NEXT YC
150 NEXT XC
160 ' *** define color ***
170 R=RND(1)*5+2:G=RND(1)*5+2:B=RND(1)*5
+2
180 FOR P=1 TO CN
190  J=P/CN:R(P)=R*J:G(P)=G*J:B(P)=B*J
200 NEXT P
210 ' *** change color ***
220 FOR K=0 TO 20
230  FOR P=1 TO CN
240   COLOR=(J+1,R(P),G(P),B(P))
250   J=(J+1) MOD CN
260  NEXT P
270  J=(J+1) MOD CN
280 NEXT K
290 GOTO 170
```

このプログラムには，まだ説明されていない命令などが使われていますが，入力して実行してみてください。

参考までに，簡単にプログラムの説明をしておきます。
まず，行70〜150で，四角形を描きます。そのために使われる変数が行10，20で定義されています。主な変数の意味は次の図のとおりです。



行170〜200では，R(P)，G(P)，B(P)という配列変数を使い，R(1)〜R(10)，G(1)〜G(10)，B(1)〜B(10)に2〜7の値を代入しています。どんな値になるかは行170のRND関数により決められます。RND関数は0以上1未満の乱数を発生する関数です。関数については第8章で詳しく説明します。
行220〜280で，R(P)，G(P)，B(P)の値を使ってカラーパレット機能によりカラーコード1〜10の色をさまざまに変化させています。

なお，プログラム中，行40，60，160，210に出てくる＇（シングルクォート）は，REMという語の省略形です。**REM**（または＇）文を使うと，プログラムの実行に関係のないコメントなどをプログラム中に書くことができます。

ページの切り換え

●ページについて

●ディスプレイページとアクティブページ

●ページの切り換え—SET PAGE

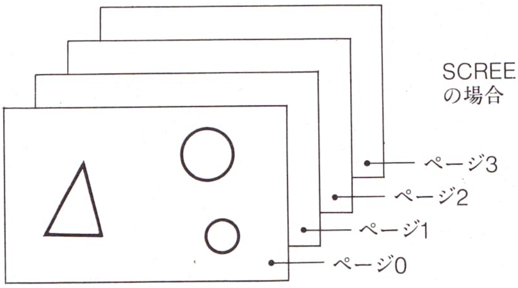
グラフィックモードの画面とページ

もう1度，55ページの画面のモードの表をみてください。その中に，「ページ」という欄があります。「ページ」が使えるのは，SCREEN 5～SCREEN 8およびSCREEN 10～SCREEN 12のグラフィックモードで，その部分だけ抜き出すと次のようになります。

モード	ページ
SCREEN 5	4
SCREEN 6	4
SCREEN 7	2
SCREEN 8	2
SCREEN 10	2
SCREEN 11	2
SCREEN 12	2

ページとは，文字どおりノートのページのようなもので，たとえばSCREEN 5，SCREEN 6のモードでは4ページ使えます。

この場合，グラフィック文で画面に図形を描いても，それは2ページあるいは4ページのうちの1ページに描いただけで，残りのページはまだ何も描かれずに残っています。



SCREEN 5，SCREEN 6
の場合

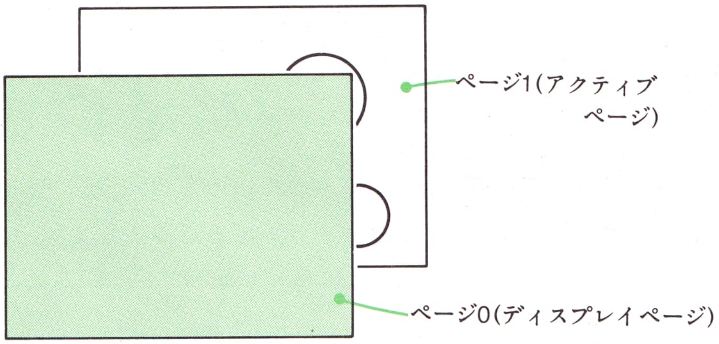
この図のように，4ページある場合は，それぞれにページ番号をつけ，それぞれページ0，ページ1，ページ2，ページ3と呼びます。

ディスプレイページとアクティブページ

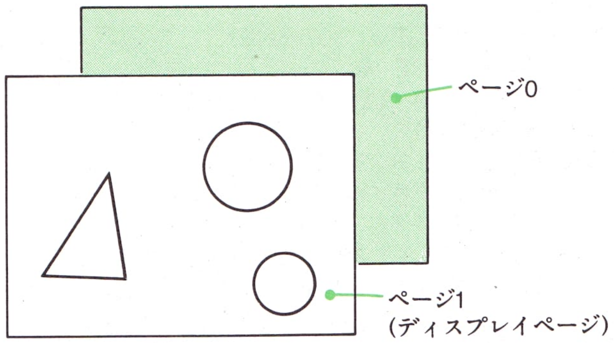
2ページ，あるいは4ページが使えるモードでも，BASICがスタートした状態では，絵を描くことができるのも，モニターテレビの画面上に見えているのもページ0です。絵を描くことができるページをアクティブページ，モニターテレビで見ることができるページをディスプレイページと呼びます。

ページ切り換えによる効果

特に指定しない限り，ディスプレイページ，アクティブページともページ0です。つまり，LINE文やCIRCLE文，DRAW文などを実行すると，ページ0に図形が描かれ，それがそのままモニターテレビに表示されます。
ところが，ディスプレイページはページ0にしておいてアクティブページをページ1に切り換えてグラフィック文を実行すると，図形はページ1に描かれますが，モニターテレビに見えているのはページ0なので，図形は画面上には表示されません。



そして，ページ1に図形が描かれたところでページ1をディスプレイページに切り換えると，できあがった図形をモニターテレビの画面で見ることができるようになります。



ページを切り換えるには　SET PAGE

ページ切り換えを行うのがSET PAGE文です。

SET PAGE（セット ページ）　ディスプレイページ，アクティブページ

SET PAGE文は，指定した番号のページをそれぞれディスプレイページ，アクティブページとします。

たとえば，ディスプレイページを0，アクティブページを1にするときは，

```
SET PAGE 0,1
```

を実行します。

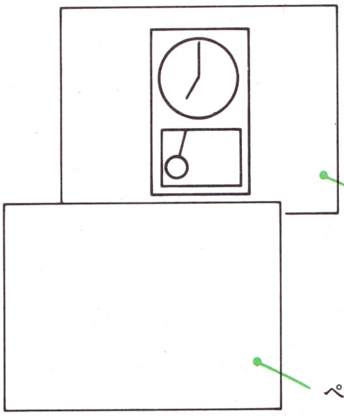
このSET PAGE文実行後は，見えるのはページ0で，図形が描かれるのはページ1になります。

ページ切り換えの実際

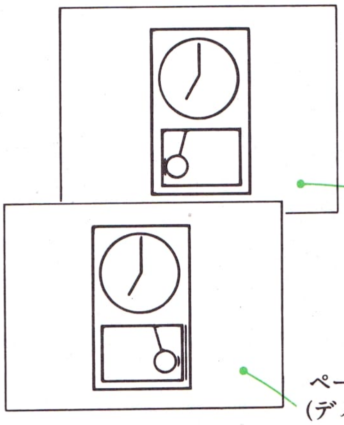
ページの切り換えを，実際のプログラムでみてみましょう。

```
10 SCREEN 5
20 SET PAGE 0,1:CLS
30 X1=110:X2=100
40 GOSUB 130
50 SET PAGE 0,0:CLS
60 X1=140:X2=150
70 GOSUB 130
80 SET PAGE 1
90 FOR T=0 TO 300:NEXT T
100 SET PAGE 0
110 FOR T=0 TO 300:NEXT T
120 GOTO 80
130 COLOR 13,3,3:CLS
140 LINE (75,10)-(175,200),,B
150 PAINT (76,11)
160 CIRCLE (125,60),40,15
170 PAINT (125,60),15
180 LINE (125,60)-(125,25),1
190 LINE (125,60)-(115,80),1
200 LINE (80,120)-(170,190),4,BF
210 LINE (X1,120)-(X2,170),10
220 CIRCLE (X2,170),12,10
230 PAINT (X2,171),10
240 RETURN
```

このプログラムは，SCREEN 5のページ0，1に異なる図形を描き，それを連続して切り換えています。
まず，ディスプレイページを0，アクティブページを1にし（行20），行130からのサブルーチンで次のような図形を描いています。



続いて，ディスプレイページ，アクティブページとも0にし（行50），次のような図形を描きます。



このとき，ページ1の図形はそのまま残っています。
そして，行80から行120までのループで，ディスプレイページを1，0，1，0と連続して切り換えているので，画面上では2つの図形が交互にあらわれ，ちょうど時計の振子が左右に振れているように見えます。
なお，行80，行100のSET PAGE文では，アクティブページの指定を省略してあります。指定を省略すると，その前に指定された状態のまま変わりません。

図形データの転送

●転送とは―COPY
●画面間の転送
●画面と内部メモリー間の転送
●画面とディスクファイル間の転送
●メモリーとディスクファイル間の転送
●ロジカルオペレーション

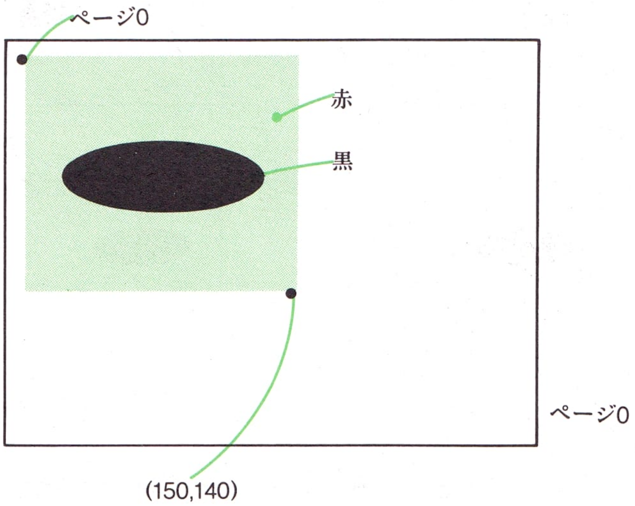
転送とは

SCREEN 5～SCREEN 8およびSCREEN 10～SCREEN 12のグラフィックモードでは，描いた図形を転送することができます。
転送とは，画面上のある大きさの領域を指定し，その領域内の1ドットごとの色の情報をそっくり他の場所へ移しかえることをいいます。
「他の場所」とは，次の3つのうちのいずれかです。
●画面（VRAM）
●内部メモリー（配列変数）
●フロッピーディスク（ファイル）

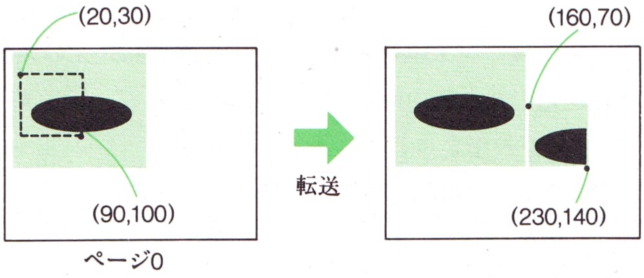
この転送を行うのがCOPY文です。

画面間の転送　COPY その1

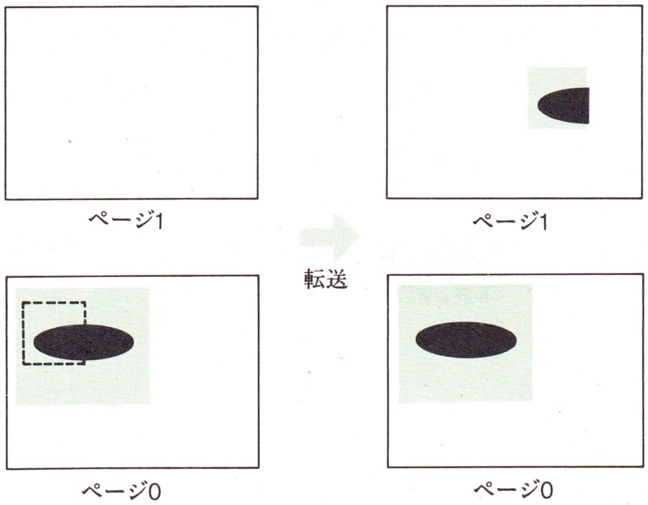
画面間の転送では，同一ページ上での転送および異なるページ間での転送が可能です。いずれの場合も，図形のデータはコンピューターのVRAM内で転送されます。今，SCREEN 5のモードのページ0に次のような図形が描かれているとします。



この図形のうち，(20，30）と（90，100）で囲まれる領域（下図の点線内）を，同じページ上の(160，70)を左上とする領域に転送すると，次のような結果が得られます。



また，別のページに転送することもできます。上と同じ図形を，ページ1の同じ位置に転送したときの結果は次のようになります。



転送を行うと，転送元のフォアグラウンドの図形，色がそのまま転送先に移動します。転送元の図形もそのまま残ります。
画面間で図形を転送するときは**COPY**文を次のように使います。

COPY（X1，Y1）－（X2，Y2）［，転送元ページ］TO（X3，Y3）［，転送先ページ］

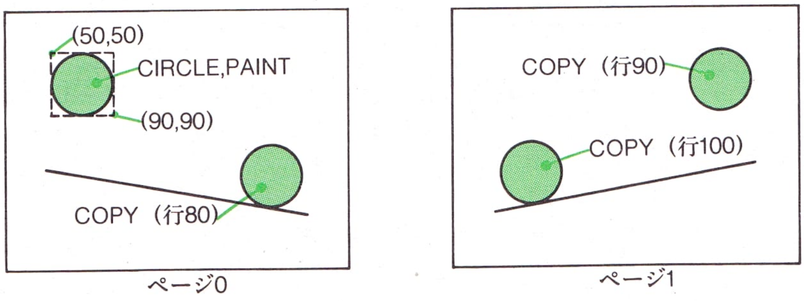
（X1，Y1）は転送元の領域の左上隅の座標，（X2，Y2）は転送元の領域の右下隅の座標，（X3，Y3）は転送先の左上隅の座標です。転送元ページ，転送先ページは，省略するとアクティブページが指定されたことになります。
それではプログラムを作って確認しましょう。

```
10 SCREEN 5
20 SET PAGE 0,0
30 LINE (10,10)-(150,140),8,BF
40 CIRCLE (90,60),40,1,,,.3
50 PAINT (90,60),1
60 COPY (20,30)-(90,100),0 TO (160,70),0
70 GOTO 70
```

行60で，ページ0の（20，30）-（90，100）の領域を，ページ0の（160，70）を左上隅とする領域に転送しています。

```
10 SCREEN 5
20 SET PAGE 0,1:CLS
30 LINE (70,165)-(180,140),1
40 SET PAGE 1,0:CLS
50 LINE (70,140)-(180,165),1
60 CIRCLE (70,70),20,12
70 PAINT (70,70),12
80 COPY (50,50)-(90,90),0 TO (160,120),0
90 COPY (50,50)-(90,90),0 TO (160,50),1
100 COPY (50,50)-(90,90),0 TO (50,120),1
110 SET PAGE 0
120 FOR T=0 TO 300:NEXT T
130 SET PAGE 1
140 FOR T=0 TO 300:NEXT T
150 GOTO 110
```

このプログラムでは，SCREEN 5のページ0，1にそれぞれ次のような図形を描くのですが，円の部分はページ0に1つCIRCLE文とPAINT文で描き（行60，70），あとはすべてCOPY文で転送して描いています。（行80，90，100）

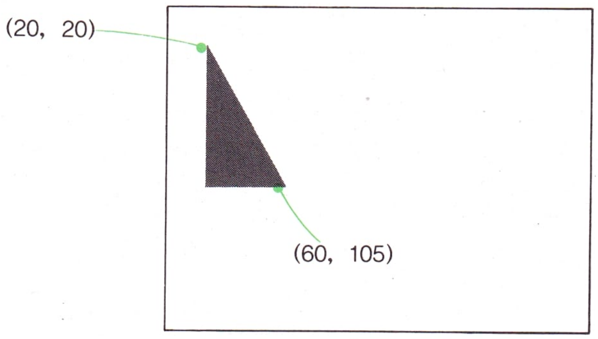


画面と内部メモリー間の転送　COPY その2

画面のデータはVRAMに記憶されているのですが，これをいったん内部メモリー(RAM)内に転送することもできます。また，内部メモリーに転送したデータを再び画面(VRAM)に転送して表示し直すことができます。このとき，表示の向きを変えることも可能です。

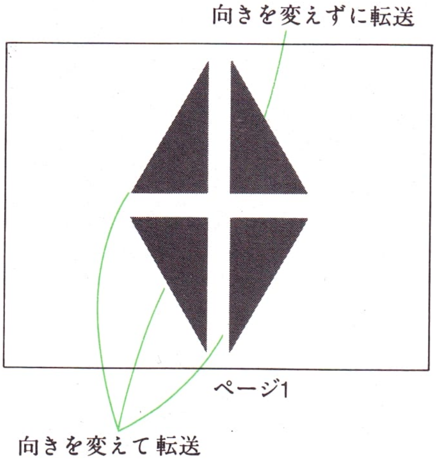
画面からメモリーへ

たとえば，SCREEN 5のページ0に次のような図形が描かれているとします。



ページ0

この画面のうち，(20，20)－(60，105)の領域をCOPY文でメモリー内にいったん転送します。そして，メモリーからページ1に，次のように転送することができます。



また，図形のデータをメモリーに転送するときは，あらかじめDIM文により数値型の配列変数を準備しておかなくてはなりません。
図形のデータをメモリーに転送するときは，**COPY**文を次のように使います。

COPY（X1，Y1）－(X2，Y2)［,転送元ページ］TO 配列変数名

（X1，Y1）は転送したい領域の左上隅の座標，（X2，Y2）は右下隅の座標です。

配列の大きさの求めかた

また，配列変数の大きさは次の式で求められます。

INT ((((ABS (X1－X2)＋1) ＊ (ABS (Y1－Y2)＋1) ＊ピクセルサイズ＋7)／8＋4)／8)＋1

INTおよびABSはBASICの関数です。関数については第8章で詳しく説明します。ピクセルサイズは，画面のモードによって異なります。

モード	ピクセルサイズ
SCREEN 5	4
SCREEN 6	2
SCREEN 7	4
SCREEN 8	8
SCREEN 10	8
SCREEN 11	8
SCREEN 12	8

たとえば，SCREEN 5のモードで（20，20）－(60，105）の領域のデータを転送するのにメモリーに用意しておかなければならない配列変数の大きさは，X1が20，Y1が20，X2が60，Y2が105ですから，X1－X2＝－40，Y1－Y2＝－85となり，次の式で求められます。

```
INT((((ABS(-40)+1*(ABS(-85)+1)*4+7)/8+4)/8)+1
```

したがって，プログラムで配列変数名をPとして準備する場合，次のような2行をプログラムの最初に書くと良いでしょう。

```
S=INT((((ABS(-40)+1)*(ABS(-85)+1)*4+7)/8+4)/8)+1
DIM P(S)
```

このように配列変数を準備しておけば，あとは次のCOPY文により(20，20)－(60，105)の領域のデータを，メモリーすなわち配列変数内に転送することができます。

```
COPY (20,20)-(60,105),0 TO P
```

COPY文では，このように配列変数名（P）だけを指定すれば良いのです。

メモリーから画面へ

次に，メモリー（配列変数）のデータを画面（VRAM）に転送する場合は，COPY文を次のように使います。

COPY　配列変数名［,向き］TO（X3，Y3)，転送先ページ

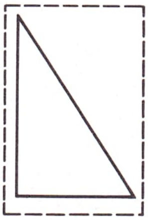
（X3，Y3）は，転送先の画面で転送されたデータを描き始める位置です。

「向き」は4とおりあり，0～3の番号で指定します。「向き」は，描き始めの位置からどの方向に向かって描くかを指示します。

「向き」の番号	描く方向
0	左上から右下 ↘
1	右上から左下 ↙
2	左下から右上 ↗
3	右下から左上 ↖

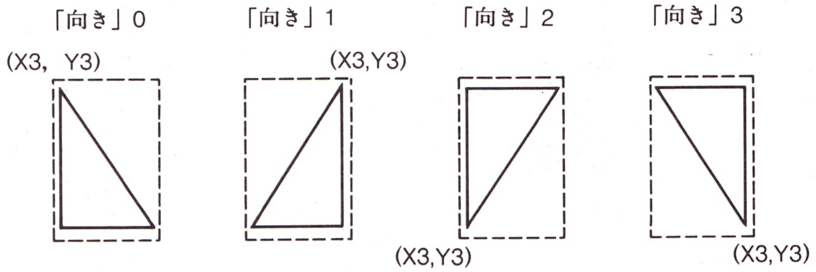
省略すると0が指定される

たとえば，次のような図形のデータがメモリーに転送されているとします。



点線内がメモリーに転送されている

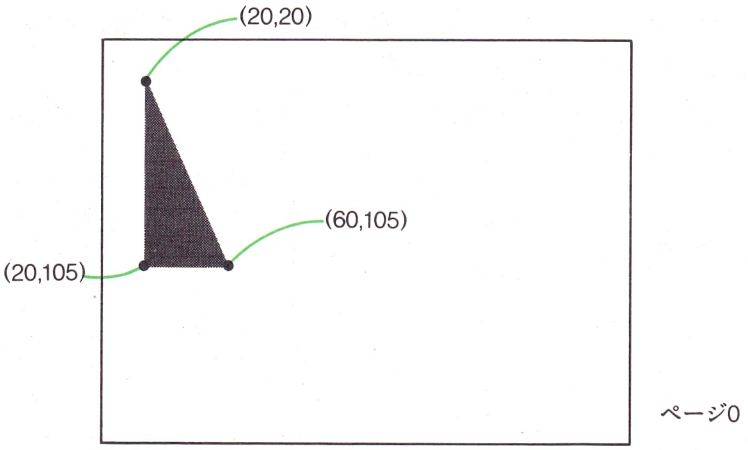
このデータを，それぞれの「向き」で画面に転送したときの (X3，Y3)，つまり描き始めの位置と図形の向きは次のようになります。



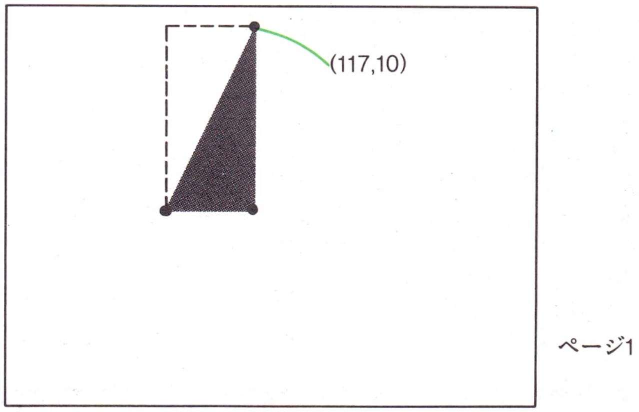
「向き」の番号を省略すると，0，すなわち「左上から右下」が選ばれます。
それでは，94ページの例を実行するプログラムを作ってみます。

```
10 S=INT((((ABS(-40)+1)*(ABS(-85)+1)*4+7
)/8+4)/8)+1
20 DIM P(S)
30 SCREEN 5
40 SET PAGE 0,1:CLS
50 SET PAGE 0,0:CLS
60 LINE (20,20)-(20,105)
70 LINE (20,20)-(60,105)
80 LINE (20,105)-(60,105)
90 PAINT (25,50)
100 COPY (20,20)-(60,105),0 TO P
110 COPY P,0 TO (138,10),1
120 COPY P,1 TO (117,10),1
130 COPY P,2 TO (138,200),1
140 COPY P,3 TO (117,200),1
150 SET PAGE 1
160 GOTO 160
```

行10，20で配列変数を準備しています。配列変数名はPです。行30～90で，SCREEN 5のページ0に次の図形を描いています。



この図形を含む，(20，20)－(60，105) の領域を，メモリー (配列変数) に転送しているのが行100のCOPY文です。そして，いったんメモリーに転送した図形データをページ1に向きを変えて転送しているのが行110～140です。
たとえば，行120では「向き」1，描き始めの座標 (117，10) で転送していますから，転送の結果は次のようになります。



他のCOPY文も同様の働きをしますから，結果として94ページのような図形がページ1に描かれることになります。

画面とフロッピーディスク間の転送　COPY その3

同じCOPY文を使って，画面 (VRAM) に描いた図形のデータを，フロッピーディスクのファイルに転送することもできます。この場合，フロッピーディスクドライブが必要です。

画面からディスクへ

画面のデータをフロッピーディスクに転送（セーブ）するためのCOPY文は次のとおりです。

```
COPY (X1, Y1) − (X2, Y2) [,転送元ページ] TO "ファイル名 [.タイプ名]"
```

ファイル名のつけ方のルールは，プログラムをセーブするときとまったく同じです。タイプ名は省略してもかまいませんが，ファイルの種類がすぐに分かるようにつけておく方が良いでしょう。この本では，以下のプログラム例でPICというタイプ名をつけることにします。

それでは，先ほどメモリーに転送したのと同じ図形をフロッピーディスクに転送するプログラムを作ってみます。

```
10 SCREEN 5
20 SET PAGE 0,0:CLS
30 LINE (20,20)-(20,105)
40 LINE (20,20)-(60,105)
50 LINE (20,105)-(60,105)
60 PAINT (25,50)
70 COPY (20,20)-(60,105),0 TO "TRIANGLE.
PIC"
```

行60までで三角形を描いています。

描いた三角形を含む領域 (20, 20) − (60, 105) のデータをフロッピーディスクのファイルに転送しているのが行70です。ファイル名はTRIANGLE, タイプ名はPICとしています。

このプログラムを実行すると，まずSCREEN 5のページ0に三角形が描かれます。三角形が描かれるとすぐにディスクドライブが作動します。このとき，図形のデータがフロッピーディスクに転送されているのです。

転送が終了すると，プログラムも終了し，画面にはOkがあらわれます。

ここで，FILES命令を実行すると，

```
TRIANGLE.PIC
```

というファイル名，タイプ名が表示され，たしかに図形のデータがフロッピーディスクのファイルに転送されたことが確認できます。

ディスクから画面へ

次は，フロッピーディスクのファイルの図形データを画面(VRAM)に転送するためのCOPY文です。

COPY ”ファイル名[.タイプ名]” [,向き] TO (X3，Y3) [,転送先ページ]

ファイル名，タイプ名は，フロッピーディスクに転送したときに使ったものと同じものを指定します。「向き」および描き始めの座標(X3，Y3)は，メモリーからの転送の場合とまったく同じです。

それでは，今フロッピーディスクに転送した図形データを，向きを4とおりに変えて画面に転送するプログラムです。

```
10 SCREEN 5
20 SET PAGE 1,1:CLS
30 COPY "TRIANGLE.PIC",0 TO (138,10),1
40 COPY "TRIANGLE.PIC",1 TO (117,10),1
50 COPY "TRIANGLE.PIC",2 TO (138,200),1
60 COPY "TRIANGLE.PIC",3 TO (117,200),1
70 GOTO 70
```

行30～60のCOPY文で，描き始めの位置および向きを変えてSCREEN 5のページ1に転送しています。

このプログラムを実行すると，データの転送される様子がよく分かります。

このプログラムでは，ディスプレイページ，アクティブページとも1にしましたが，ディスプレイページとアクティブページを異なるページにしておいて，フロッピーディスクからの画像データをアクティブページに転送し，転送し終わったらそのページをディスプレイページに切り換えれば，転送終了後の図形をいきなり表示させることになります。

さきほどのプログラムを次のように変更して実行してみてください。

```
10 SCREEN 5
20 SET PAGE 0,1:CLS
30 COPY "TRIANGLE.PIC",0 TO (138,10),1
40 COPY "TRIANGLE.PIC",1 TO (117,10),1
50 COPY "TRIANGLE.PIC",2 TO (138,200),1
60 COPY "TRIANGLE.PIC",3 TO (117,200),1
65 SET PAGE 1
70 GOTO 70
```

フロッピーディスクとメモリー間の転送　COPY その4

COPY文でフロッピーディスクにセーブした図形データを，メモリー(配列変数)に転送することもできます。COPY文を次の形で使います。

```
COPY "ファイル名 [.タイプ名]" TO 配列変数名
```

また，メモリー（配列変数）に転送しておいた図形データをさらにフロッピーディスクに転送するときは，COPY文を次の形で使います。

```
COPY 配列変数名 TO "ファイル名 [.タイプ名]"
```

ロジカルオペレーション

COPY文で図形のデータを転送するときに，転送する図形の色のカラーコードと，転送先の画面の色のカラーコードとの間でロジカルオペレーション（論理演算）を行うことができます。

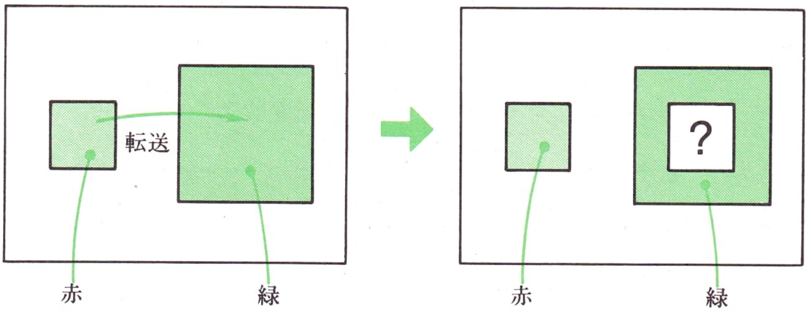
COPY文でできるロジカルオペレーションは次の10種類です。

PSET，PRESET，XOR，OR，AND，TPSET，TPRESET，TXOR，TOR，TAND

ロジカルオペレーションの結果がどうなるかを知るには，カラーコードを2進数に変換しておくとすぐに分かります。SCREEN 5のモードでの，BASICスタート時の16色のカラーコードを2進数で表わすと次のようになります。

色	カラーコード(10進数)	カラーコード(2進数)	色	カラーコード(10進数)	カラーコード(2進数)
透明	0	0000	赤	8	1000
黒	1	0001	淡い赤	9	1001
緑	2	0010	濃い黄	10	1010
淡い緑	3	0011	淡い黄	11	1011
濃い青	4	0100	濃い緑	12	1100
淡い青	5	0101	マゼンタ	13	1101
濃い赤	6	0110	灰	14	1110
空色	7	0111	白	15	1111

ここで，次のように赤（カラーコード1000）の図形を緑（カラーコード0010）の上に転送するとします。



ロジカルオペレーションを行わなければ，緑の四角形の中に赤の四角形が描かれることになります。しかし，ここでロジカルオペレーションを行うと，緑と赤が重なる ? の部分の色は，そのロジカルオペレーションの種類によっていろいろな結果になるのです。

ロジカルオペレーションORの場合

ロジカルオペレーションのひとつ，ORを例にしてみましょう。ORとは，2つのカラーコード（2進数）の各けた（0か1）に対して，次のような演算を行うものです。

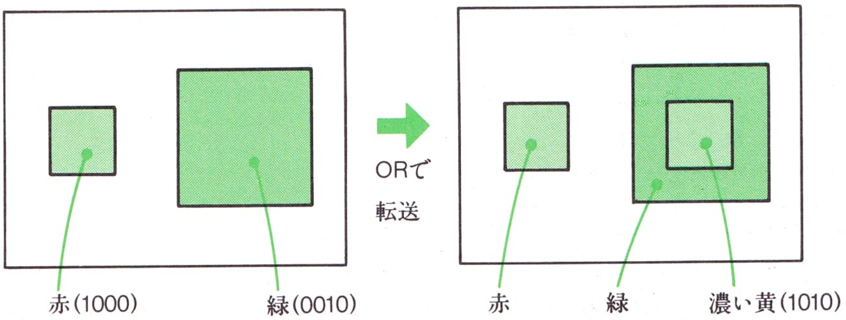
X	Y	XとYのORの結果
0	0	0
0	1	1
1	0	1
1	1	1

上の例では，赤のカラーコード1000と緑のカラーコード0010のそれぞれのけたどうし，ORの演算を行います。

赤……1000
緑……0010

OR… 1010

ORの結果は1010となりました。ここで再びさきほどのカラーコードの表から，カラーコード1010の色を探すと，濃い黄となっています。
したがって，緑の上に赤の図形をORというロジカルオペレーションを行って転送すると，重なった部分は濃い黄になるのです。



OR以外のロジカルオペレーションの働きは次のとおりです。

●PSET—転送する色が，転送先の色にかかわらず，そのままの色で表示されます。(とくにロジカルオペレーションは行いません。)

●PRESET—転送する色のカラーコードの各けたの0と1を反転させます。転送先の色には関係ありません。たとえば，赤(1000)をPRESETで転送すると，空色(0111)になります。

●XOR—転送する色のカラーコードと転送先の色のカラーコードとの間で次の演算を行います。

X	Y	XとYの XORの結果
0	0	0
0	1	1
1	0	1
1	1	0

たとえば，赤（1000）と緑（0010）の間でXORを行うと，結果は濃い黄（1010）となります。

●AND—転送する色のカラーコードと転送先の色のカラーコードとの間で次の演算を行います。

X	Y	XとYの ANDの結果
0	0	0
0	1	0
1	0	0
1	1	1

たとえば，赤（1000）と緑（0010）の間でANDを行うと，結果は透明（0000）になります。

●TPSET，TPRESET，TOR，TXOR，TAND—それぞれ，PSET，PRESET，OR，XOR，ANDと同じですが，Tをつけると転送する色に透明がある場合，転送先の色にかかわらず転送後もその部分だけは透明になります。

ロジカルオペレーションを行うCOPY文

ロジカルオペレーションを行うときは，COPY文を次のように使います。

●画面から画面へ

COPY (X1，Y1)－(X2，Y2)，転送元ページ　TO (X3，Y3)，転送先ページ，ロジカルオペレーション

例：COPY (10，10)－(100，100)，0　TO (30，30)，1，XOR

●メモリー（配列変数）から画面へ

COPY 配列変数名，向き　TO (X3，Y3)，転送先ページ，ロジカルオペレーション

例：COPY P，1 TO (30，30)，0，TAND

●フロッピーディスク（ファイル）から画面へ

COPY "ファイル名.タイプ名"，向き　TO (X3，Y3)，転送先ページ，ロジカルオペレーション

例：COPY "TRIANGLE. PIC"，2 TO (30，30)，1，PRESET

ロジカルオペレーションを省略すると，PSETが指定されたのと同じになります。
次に，画面から画面へ図形を転送するときにロジカルオペレーションを行うプログラムを示します。

```
10 SCREEN 5
20 SET PAGE 0,0:CLS
30 READ C1,C2,C3,C4
40 LINE (30,30)-(100,100),C1,BF
50 LINE (45,45)-(85,85),C2,BF
60 LINE (126,105)-(188,210),C3,BF
70 LINE (189,105)-(250,210),C4,BF
80 COPY (30,30)-(100,100),0 TO (153,124)
,0,OR
90 GOTO 90
100 DATA 8,3,10,2
```

次のプログラム例では，行40〜90で描いた図形を配列変数Sに転送したあと，行120〜150で別の図形を描き，その上に配列変数から10種類のロジカルオペレーションにより次々に最初の図形を転送し直しています。各ロジカルオペレーションの効果の違いが良く分かります。

```
10 COLOR 15,0,0 : SCREEN 5
20 DIM S(170)
30 '
40 XC=100:YC=100:R=5
50 FOR I=0 TO 15
60  J=I*4+3
70  LINE(XC+J,YC+5)-STEP(3,7),I,BF
80 NEXT
90 LINE (XC,YC)-STEP(4*20,17),15,B
100 COPY (XC,YC)-STEP(4*20,17) TO S
110 '
120 COLOR 15,4,1:CLS
130 FOR L=0 TO 100 STEP 2
140  W=L*2:LINE(128-L,106-L)-STEP(W,W),8
,B
150 NEXT
160 COPY S TO (50,  5),,PSET
170 COPY S TO (50, 25),,PRESET
180 COPY S TO (50, 45),,XOR
190 COPY S TO (50, 65),,OR
200 COPY S TO (50, 85),,AND
210 COPY S TO (50,105),,TPSET
220 COPY S TO (50,125),,TPRESET
230 COPY S TO (50,145),,TXOR
240 COPY S TO (50,165),,TOR
250 COPY S TO (50,185),,TAND
260 GOTO 260
```

SCREEN文でできること

- SCREEN文で設定できること
- キー音を消す
- カセットテープレコーダーとのボーレート設定
- プリンターのタイプ
- インターレースモード

SCREEN文で設定できること

SCREEN文は，画面のモード以外にも各種モードを設定するのに使います。

> SCREEN[画面モード]，[スプライトの大きさ]，[キークリックスイッチ]，[ボーレート]，[プリンタータイプ]，[インターレースモード]

スプライトの大きさについては，次章で説明します。

キークリックスイッチ，ボーレート，プリンタータイプ

キークリック音を消す

キーボードのキーを押すときに聞こえる音がキークリック音です。この音が出るようにするか，出ないようにするかを，SCREEN文の3番目のパラメーター(キークリックスイッチ）で決めることができます。

キークリックスイッチ0で音が出なくなり，キークリックスイッチ0以外（1～255）で音が出るようになります。

```
SCREEN ,,0--- ----キークリック音を消す
SCREEN ,,1--------キークリック音を出す
```

カセットボーレートの設定

ボーレートとは,1秒間に転送されるデータの量をビット単位で示したものです。1200ボーなら1秒間に1200ビット,2400ボーなら1秒間に2400ビットのデータが転送されます。

SCREEN文の4番目のパラメーター(ボーレート)は，カセットテープレコーダーとのデータ転送時のボーレートを設定します。ボーレート1で1200ボー，ボーレート2で2400ボーになります。初期状態は1（1200ボー）です。

```
SCREEN ,,,1--------1200ボーにする
SCREEN ,,,2--------2400ボーにする
```

カセットテープにプログラムなどをセーブする前に,SCREEN,,,2により2400ボーに設定しておいた場合は，このテープからプログラムなどをロードするときも，前もってSCREEN,,,2により2400ボーにしておかなければなりません。

プリンタータイプ

SCREEN文の5番目のパラメーター(プリンタータイプ)は，使用するプリンターに合わせて設定します。MSX仕様のプリンターを使う場合,プリンタータイプを0にします。MSX仕様ではないプリンターを使う場合,プリンタータイプを0以外(1~255)にします。初期状態は0（MSX仕様のプリンター）になっています。

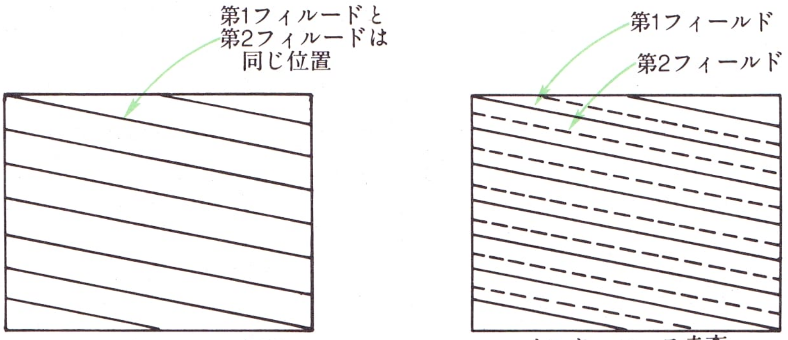
MSX仕様のプリンターとは，MSXのグラフィックキャラクターとひらがなのフォントを内蔵し，MSXパーソナルコンピューターに対応したプリンターです。

MSX仕様ではないプリンターを使う場合,SCREEN,,,,1を実行しておくと,グラフィックキャラクターはスペースに，ひらがなはカタカナに変換されて印字されます。

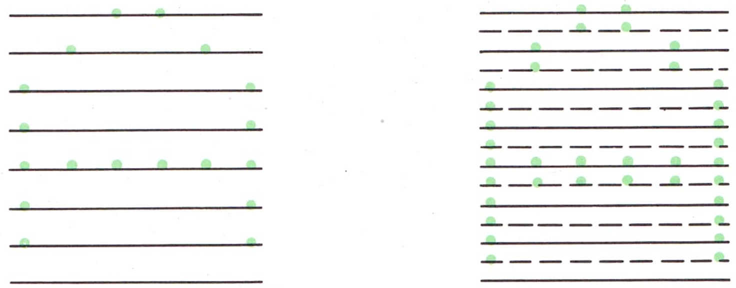
インターレースモード

インターレース走査について

MSXコンピューターでは，通常ノンインターレース走査を行います。
SCREEN文の6番目のパラメーター（インターレースモード）を使うと，これをインターレース走査に切り換えることができます。
インターレース走査にすると，第1フィールドの走査位置と第2フィールドの走査位置がずれて，よりきめの細かい表示が得られます。ただし，ふつうのテレビやモニターを使った場合，かえって画面にちらつきが生じます。



ノンインターレース走査　　インターレース走査



ノンインターレース走査の場合　　インターレース走査の場合

偶数／奇数ページ切り換え表示モードについて

SCREEN文のインターレースモードのパラメーターにより，偶数／奇数ページ切り換えモードを選択することもできます。このモードにするときは，まずSET PAGE文でディスプレイページを奇数ページ(1または3)にしておきます。そして，SCREEN文により偶数／奇数ページ切り換え表示モードにすると，ディスプレイページとディスプレイページよりひとつ小さい番号のページとが高速で交互に入れ換わって表示されます。

インターレースモードのパラメーターの値は0～3で，各値の意味は次のとおりです。

インターレースモード	設定されるモード
0	通常のモード（ノンインターレース，ページ切り換えなし）
1	インターレースモード
2	偶数／奇数ページ切り換え表示モード
3	偶数／奇数ページ切り換え，インターレースモード

次のプログラムは，SCREEN 5のページ0に黄色の円，ページ1に白のだ円を描き，まず，SET PAGE文によるディスプレイページの切り換えによりページ0，1を交互に表示させます。このとき，ページ切り換えの間隔をだんだん短くしています。そして，最後に偶数／奇数ページ切り換え表示モードにして，おもしろい効果を得ています。

```
10 COLOR 15,4,4:SCREEN 5,,,,,0
20 SET PAGE 0,1:CLS
30 XC=128 : YC=100
40 '*** draw yellow circle ***
50 SET PAGE 0,0
60 CIRCLE (XC,YC),45,10
70 PAINT (XC,YC),10
80 FOR T=0 TO 2000:NEXT T
90 '*** draw white oval ***
100 SET PAGE 1,1
110 CIRCLE (XC,YC),90,15,,,.7
120 PAINT (XC,YC),15
130 FOR T=0 TO 2000:NEXT T
140 '*** change pages ***
150 J=1200:DP=0:AP=1
160 SET PAGE DP,DP:SWAP DP,AP
170 FOR T=0 TO J:NEXT T
180 J=J*.8:IF J>1 THEN 160
190 '*** even/odd mode ***
200 SET PAGE 1,1
210 SCREEN ,,,,,2
220 GOTO 220
```

第5章
スプライトを動かそう

スプライトの定義と動かしかた

- スプライトとは
- スプライトパターンの定義——SPRITE$
- スプライトを表示する——PUT SPRITE
- スプライトを動かす

スプライトとは

スプライトとは，8×8ドットまたは16×16ドットの大きさで，自由に形を定義して画面上を動かすことのできるパターンのことです。

MSX$_{2+}$-BASICでは，スプライトパターンを32面あるスプライト面のどれかに表示させ，動かすことができます。

画面モードとスプライト

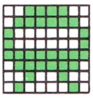
スプライトを使うことができるのは，SCREEN 0以外のモード，すなわち**SCREEN 1～SCREEN 8**および**SCREEN 10～SCREEN 12**のモードです。このうち，SCREEN 4～SCREEN 8とSCREEN 10～SCREEN 12のモードでは，スプライトの表示に関してSCREEN 1～SCREEN 3のモードではできないいくつかの機能を使うことができます。

モード	画面	スプライト
テキストモード	SCREEN 1	スプライト使用可
グラフィックモード	SCREEN 2	スプライト使用可
グラフィックモード	SCREEN 3	スプライト使用可
グラフィックモード	SCREEN 4	機能が強化されたスプライト使用可
グラフィックモード	SCREEN 5	機能が強化されたスプライト使用可
グラフィックモード	SCREEN 6	機能が強化されたスプライト使用可
グラフィックモード	SCREEN 7	機能が強化されたスプライト使用可
グラフィックモード	SCREEN 8	機能が強化されたスプライト使用可
グラフィックモード	SCREEN 10	機能が強化されたスプライト使用可
グラフィックモード	SCREEN 11	機能が強化されたスプライト使用可
グラフィックモード	SCREEN 12	機能が強化されたスプライト使用可

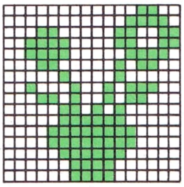
スプライトパターンの種類

ひとつのスプライトパターンは8×8ドットまたは16×16ドットで構成され，それぞれに標準と拡大の2つの大きさがあります。拡大は，標準の大きさのたて，横を2倍にしたものです。

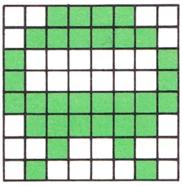
8×8ドット
標準



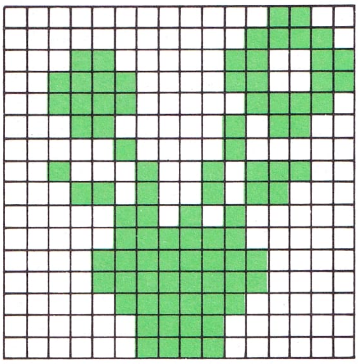
16×16ドット
標準



8×8ドット
拡大



16×16ドット
拡大



スプライトの大きさの指定――SCREEN文

どの大きさのスプライトパターンにするかを決めるのもSCREEN文です。SCREEN文の2番目のパラメーターがスプライトの大きさを選択します。

SCREEN　モード，スプライトの大きさ

スプライトの大きさ	選ばれる大きさ
0	8×8 ドット標準
1	8×8 ドット拡大
2	16×16ドット標準
3	16×16ドット拡大

たとえば，

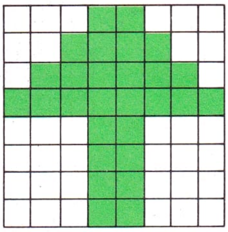
```
SCREEN 2,3
```

は，SCREEN 2のモードで16×16ドット拡大のスプライトを使用する，と指定する文です。1度SCREEN文でスプライトの大きさを指定すると，すべてのスプライト面に表示されるスプライトがその大きさになります。

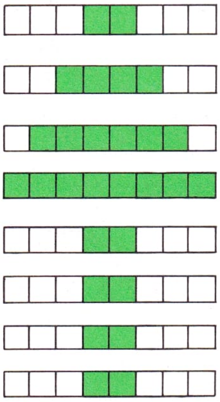
スプライトパターンの定義　SPRITE$変数

8×8ドットのスプライトパターン

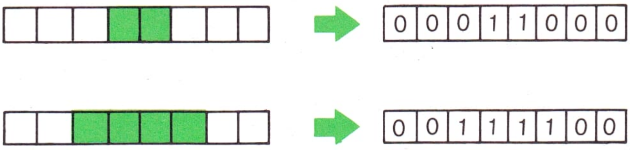
8×8ドットのパターンを定義するときは，まずパターンを横に8列に分解します。たとえば，次のような矢印のパターンを定義するとします。



このパターンを横8列に分解すると，それぞれが8つのドットからなる小さいパターンに分かれます。



次に，それぞれの列のパターンの，点を打つドットを1，打たないドットを0として並べたものを2進数の数値とします。一番上の列の場合なら00011000，次の列なら00111100という具合です。



このようにしてできた2進数を，今度は16進数（または10進数）に変換します。一番上の列は，

00011000（2進数）→18（16進数）または24（10進数）

次の列は

00111100（2進数）→3C（16進数）または60（10進数）

となります。

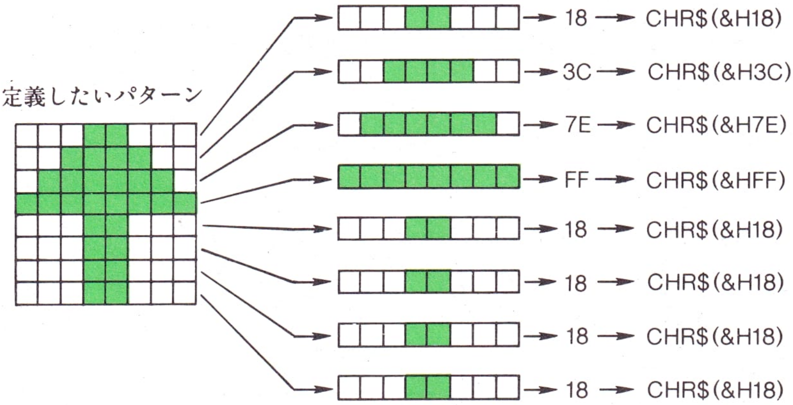
2進数を16進数に変換するのに慣れていない人は，次の表を参考にして，8ドットのパターンをさらに左4ドット，右4ドットに分けて16進数のひとけた（0からFまで）に直していくと簡単です。

パターン	16進数	パターン	16進数
□□□□	0	■□□□	8
□□□■	1	■□□■	9
□□■□	2	■□■□	A
□□■■	3	■□■■	B
□■□□	4	■■□□	C
□■□■	5	■■□■	D
□■■□	6	■■■□	E
□■■■	7	■■■■	F

あらためて □□□■■□□□ のパターンを見ると，左4ドットは □□□■，右4ドットは ■□□□ ですから，上の表から16進数18に変換されることが分かります。

最後に，これらの16進数（あるいは10進数）を文字コードとする文字を，CHR$関数によって求めます。

ここまでをまとめると，次のようになります。



8×8ドットのスプライトパターンは，このようにして求められる文字データを上から順番に加えていき，ひとつの文字列としてSPRITE$変数に代入して定義します。この例の矢印のパターンの場合，

```
SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$(&H
7E)+CHR$(&HFF)+CHR$(&H18)+CHR$(&H18)+CHR
$(&H18)+CHR$(&H18)
```

で定義されます。定義されたスプライトパターンの番号は1で，SPRITE$(1)の(　)内の数値1によって示されています。
(なお，関数については第8章で詳しく説明します。)

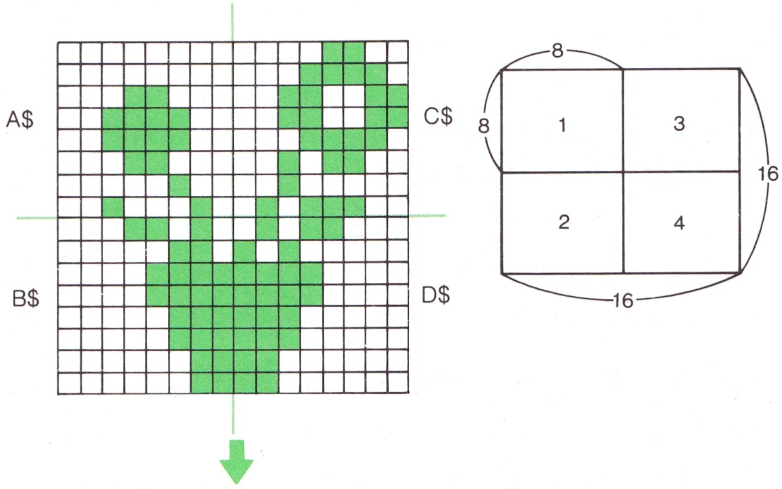
SPRITE$（スプライト番号）=文字式

なお，CHR$関数で得られる文字があれば，その文字を直接使ってもかまいません。上の例では，CHR$(&H3C)は"<"，CHR$(&H7E)は"～"ですから，次のように定義することもできます。

```
SPRITE$(1)=CHR$(&H18)+"<"+"~"+CHR$(&HFF)
+CHR$(&H18)+CHR$(&H18)+CHR$(&H18)+CHR$(&
H18)
```

16×16ドットのスプライトパターン

16×16ドットのスプライトパターンを定義するときも，要領は同じです。ただ，16×16ドットのスプライトパターンを8×8ドットのスプライトパターン4つの集まりとみなし，この4つのパターンを次のような順番でつなぎ合わせて定義します。




```
A$=CHR$(&H0)+CHR$(&H0)+CHR$(&H18)+CHR$(&
H3C)+CHR$(&H3C)+CHR$(&H18)+CHR$(&H4)+CHR
$(&H22)
B$=CHR$(&H1A)+CHR$(&H6)+CHR$(&HF)+CHR$(&
HF)+CHR$(&H7)+CHR$(&H7)+CHR$(&H3)+CHR$(&
H3)
C$=CHR$(&HC)+CHR$(&H1E)+CHR$(&H33)+CHR$(
33)+CHR$(&H1E)+CHR$(&H2C)+CHR$(&H20)+CHR
$(&H5C)
D$=CHR$(&H58)+CHR$(&HA0)+CHR$(&HF0)+CHR$
(&HF0)+CHR$(&HE0)+CHR$(&HE0)+CHR$(&HC0)+
CHR$(&HC0)
SPRITE$(2)=A$+B$+C$+D$
```

定義できるスプライトパターンの数

8×8ドットのスプライトパターンの番号は0～255まで，16×16ドットの場合0から63までの間でとることができます。つまり，8×8ドットのスプライトパターンは最大256個，16×16ドットのスプライトパターンは最大64個まで定義することができます。

スプライトパターンの表示　PUT SPRITE

定義したスプライトパターンをスプライト面に表示させるには，PUT SPRITE文を使います。

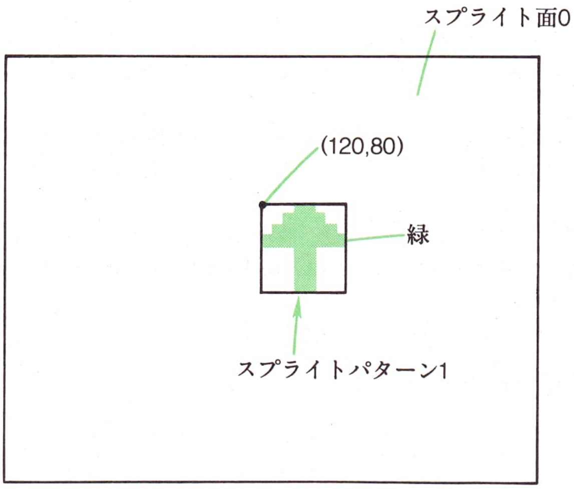
PUT SPRITE（プット スプライト）　スプライト面番号，［(X，Y)］，［カラー］，［スプライトパターン番号］

PUT SPRITE文により，指定した番号のスプライトパターンを指定したスプライト面の指定位置に，指定のカラーで表示することができます。

たとえば，上で定義したスプライトパターン（パターン番号1）を，スプライト面0の（120，80）の位置に緑（カラーコード2）で表示させるときは

```
PUT SPRITE 0,(120,80),2,1
```

を実行します。指定した表示位置は，スプライトパターンの枠の左上のドットにあたります。このX，Y座標は，グラフィック画面の（0，−1）を原点（0，0）とする座標系上で指定します。



PUT SPRITE 0,(120,80),2,1

スプライトパターン表示のルール（SCREEN 1，2，3の場合）

- ひとつのスプライト面にはひとつのスプライトパターンしか表示できません。
- 異なるスプライト面にあるスプライトパターンが重なったときは，後面(番号の大きい方)のスプライト面にあるスプライトパターンが前面のスプライトパターンにかくれるように見えます。
- 5つ以上のスプライトパターンがよこ一列に並んだときは,優先度の高い(より小さい番号のスプライト面の）スプライトパターンが4つまで表示されます。
- 表示位置の指定を省略すると，直前のグラフィック命令で指定された位置が指定されたものとみなされます。
- カラーコードを省略すると，フォアグラウンドカラーが指定されたものとみなされます。
- スプライトパターン番号を省略すると，スプライト面番号と同じ番号が指定されたものとみなされます。

スプライトを動かす

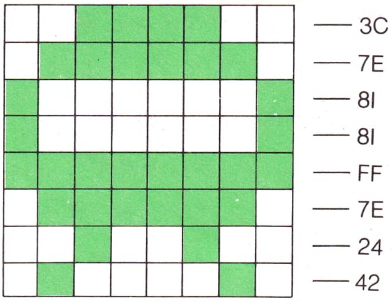
スプライトパターンを動かすには，PUT SPRITE文で指定する表示位置を変えながらPUT SPRITE文を繰り返し実行します。1度PUT SPRITE文を実行すると，同じスプライト面上にあった前のスプライトは消えるので，プログラムによっていちいち消す必要はありません。また，1ドット単位で表示位置を変えることができますから動きもなめらかです。

次のプログラムは，画面内を斜め方向にUFOのパターンを飛び回るように動かすものです。

```
10 SCREEN 2,0
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 COLOR ,1,1:CLS
40 X=120:Y=50:VX=1:VY=1
50 PUT SPRITE 0,(X,Y),5,0
60 X=X+VX
70 IF X>240 OR X<0 THEN VX=-VX
80 Y=Y+VY
90 IF Y>180 OR Y<0 THEN VY=-VY
100 GOTO 50
```

画面モードは2，スプライトサイズは8×8ドット標準です（行10）。

行20で定義されるスプライトパターンは次のとおりです。

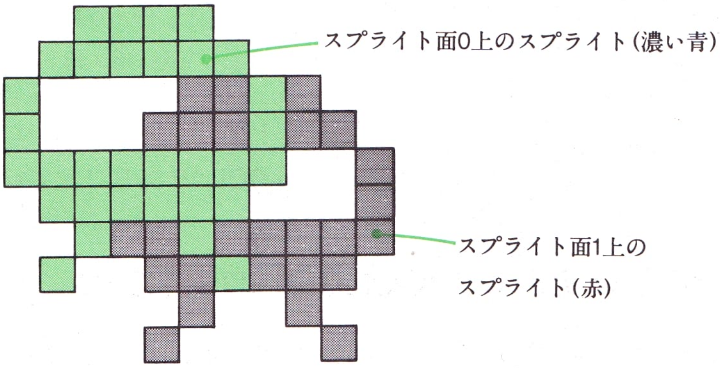


このスプライトパターンを，行50のPUT SPRITE文により表示していますが，このときの表示位置を示すX，Yの値が最初は（120，50）に設定されます（行40）。このX，Yの値を行60～90により変更しては行50のPUT SPRITE文に戻るので，画面上ではスプライトパターンが動くことになります。

次のプログラムは，スプライトの重なりをみるためのものです。

```
10 SCREEN 2,1
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 COLOR ,1,1:CLS
40 FOR X=0 TO 117
50 PUT SPRITE 0,(X,80),4,0
60 PUT SPRITE 1,(240-X,84),8,0
70 NEXT X
80 GOTO 80
```

重なりのようすをよく見るために，スプライトサイズは8×8ドット拡大にしてあります。スプライトパターンは前のプログラムのものと同じです。このスプライト(パターン番号0)を，行50と60のふたつのPUT SPRITE文で，スプライト面0と1に表示させます。スプライト面0では濃い青で，左から右へ移動させ，スプライト面1では赤で右から左へ移動させます。2つのスプライトが重なると，スプライト面0のスプライト（濃い青）が上の方に見えることが分かります。



機能強化スプライトを使う

- 強化機能について
- スプライトの色を変える——COLOR SPRITE
- 1ラインごとに色を指定——COLOR SPRITE$
- スプライト定義のテクニック

スプライト機能の強化

SCREEN 4～SCREEN 8およびSCREEN 10～SCREEN 12の各モードでは，機能の強化されたスプライトを使うことができます。

SCREEN 1～SCREEN 3のスプライトに比べてどんな機能が強化されているかをまとめると，次のようになります。

機　　能	SCREEN 1～3	SCREEN 4～8，10～12
横1列に表示できるスプライトの数	最大　4	最大　8
スプライトの色	1スプライト　1色	1スプライト 最大8色（8×8ドット） または16色(16×16ドット)
スプライトの色の変更	PUT SPRITE文で行う	PUT SPRITE文または COLOR SPRITE文で行う

スプライトの色を変える　COLOR SPRITE

スプライトの色は，PUT SPRITE文で指定しますが，SCREEN 4～8と10～12のモードでは表示後のスプライトの色をCOLOR SPRITE文で変えることができます。

COLOR SPRITE（カラー スプライト）（スプライト面番号）＝カラー

COLOR SPRITE文により，指定したスプライト面上のスプライトパターンの色が指定した色に変わります。

たとえば，スプライト面2に表示されたスプライトパターンの色を赤(カラーコード8）にするときは

```
COLOR SPRITE(2)=8
```

を実行します。

次のプログラムを実行すると，6機の異なる色のUFOが次々と着陸し，全部着陸し終わると6機とも同じ色で次々に色を換えていきます。最後に透明になり，着陸のところから繰り返します。

```
10 SCREEN 5,1
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 COLOR ,1,1:CLS
40 X1=0:Y1=118
50 FOR L=1 TO 14
60 READ X2,Y2
70 LINE (X1,Y1)-(X2,Y2),3
80 X1=X2:Y1=Y2
90 NEXT L
100 PAINT (1,119),3
110 P=0:X=5:YE=101:C=3:GOSUB 250
120 P=1:X=45:YE=122:C=4:GOSUB 250
130 P=2:X=83:YE=87:C=5:GOSUB 250
140 P=3:X=133:YE=143:C=6:GOSUB 250
150 P=4:X=168:YE=127:C=7:GOSUB 250
160 P=5:X=218:YE=96:C=8:GOSUB 250
170 FOR S=1 TO 10
180 FOR SP=0 TO 5
190 SC=S+3:IF S=10 THEN SC=0
200 COLOR SPRITE (SP)=SC
210 NEXT SP
220 FOR T=0 TO 300:NEXT T
230 NEXT S
240 GOTO 110
250 FOR Y=-8 TO YE
260 PUT SPRITE P,(X,Y),C,0
270 NEXT Y
280 RETURN
290 DATA 26,118,40,139,67,139
300 DATA 79,104,103,104,128,160
310 DATA 154,160,161,144,192,144
320 DATA 200,113,252,113,252,212
330 DATA 0,212,0,118
```

（行40〜90）背景を描く

（行110〜160）スプライトを動かす

（行170〜230）スプライトの色を変える

（行250〜280）スプライトを動かすサブルーチン

（行290〜330）背景の絵のデータ

スプライトの色を変えているのが行200のCOLOR SPRITE文です。変数SPには，スプライト面番号が代入されます。カラーコード（変数SC）は3から13まで変化しますが，ループの最後（S=10）のとき，カラーコードを0（透明）にしています。

1ラインごとに色を指定する　COLOR SPRITE$

ひとつのスプライトは，8×8ドットの場合横8ライン（列），16×16ドットの場合横16ラインから構成されます。
SCREEN 4～8のモードでは，スプライトの色を，ラインごとに指定することができます。

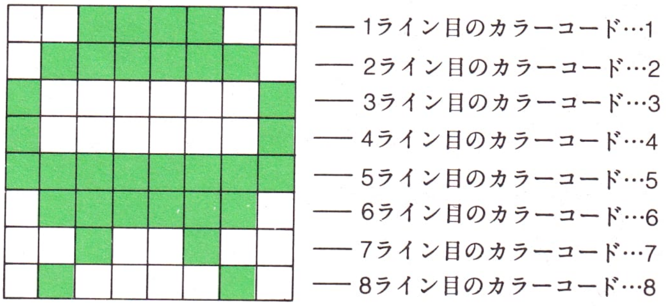
COLOR SPRITE$（スプライト面番号）＝文字式

COLOR SPRITE$文により，指定したスプライト面上のスプライトパターンの1ラインごとの色を文字式により指定することができます。

文字式は，

　CHR$（カラーコード）

を＋で加算していきます。最初のCHR$（カラーコード）が一番上のラインの色を指定します。次のCHR$（カラーコード）は二番目のラインの色を指定します。



8×8ドットのスプライトパターンの色を上の図のようにするときは，このパターンがスプライト面0上にあるとして，

```
COLOR SPRITE$(0)=CHR$(1)+CHR$(2)+CHR$(3)
+CHR$(4)+CHR$(5)+CHR$(6)+CHR$(7)+CHR$(8)
```

を実行します。
文字式のCHR$（カラーコード）が7つ以下であれば，指定されないラインの色は変わりません。

```
COLOR SPRITE(0)=CHR$(1)+CHR$(2)
```

では，1ライン目の色がカラーコード1の色，2ライン目の色がカラーコード2の色になりますが，3ライン目以下は変化しません。

次のプログラムは，先のスプライトの色を変えるプログラムの一部を修正して，赤と濃い青のしま模様のUFOが表示されるようにしたものです。

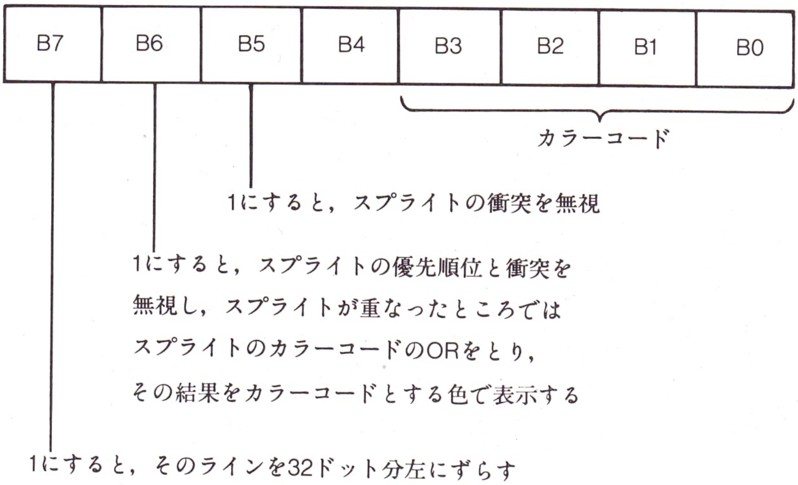
```
10 SCREEN 5,1
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 COLOR ,1,1:CLS
40 X1=0:Y1=118
50 FOR L=1 TO 14
60 READ X2,Y2
70 LINE (X1,Y1)-(X2,Y2),3
80 X1=X2:Y1=Y2
90 NEXT L
100 PAINT (1,119),3
110 P=0:X=5:YE=101:C=3:GOSUB 250
120 P=1:X=45:YE=122:C=4:GOSUB 250
130 P=2:X=83:YE=87:C=5:GOSUB 250
140 P=3:X=133:YE=143:C=6:GOSUB 250
150 P=4:X=168:YE=127:C=7:GOSUB 250
160 P=5:X=218:YE=96:C=8:GOSUB 250
170 FOR SP=0 TO 5
180 COLOR SPRITE$(SP)=CHR$(8)+CHR$(4)+CH
R$(8)+CHR$(4)+CHR$(8)+CHR$(4)+CHR$(8)+CH
R$(4)
190 NEXT SP
200 GOTO 200
250 FOR Y=-8 TO YE
260 PUT SPRITE P,(X,Y),C,0
270 NEXT Y
280 RETURN
290 DATA 26,118,40,139,67,139
300 DATA 79,104,103,104,128,160
310 DATA 154,160,161,144,192,144
320 DATA 200,113,252,113,252,212
330 DATA 0,212,0,118
```

変更（行170〜200）

変更した部分のうち行180でスプライトパターンを赤と濃い青のしま模様にしています。

文字式のデータについて

COLOR SPRITE$文の，CHR$関数で使うデータはカラーコードだけに限りません。データを2進数で表わしたとき，各ビットは次のような働きをします。



スプライトの衝突は，衝突による割り込みをかけるときに使用します。(第9章を参照してください。)

B7を1にし，カラーコード(B3～B0)を0010(＝2，緑)にすると，データは10000010となります。10000010は16進数で&H82ですから，

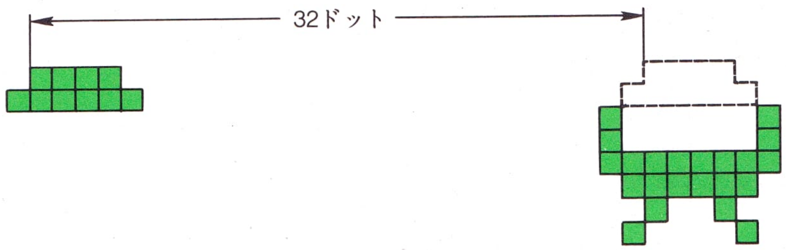
```
COLOR SPRITE$(0)=CHR$(&H82)
```

を実行すると，スプライト面0上のスプライトパターンの1ライン目が緑になるとともに左へ32ドット分ずれます。

次のプログラムを実行すると，UFOのハッチが開き，宇宙人（？）が姿を現わします。

```
10 SCREEN 5,1
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 SPRITE$(1)=CHR$(&H58)+CHR$(&H58)+CHR$
(&H7E)+CHR$(&H1A)+CHR$(&H18)+CHR$(&H18)+
CHR$(&H0)+CHR$(&H0)
40 CLS
50 PUT SPRITE 0,(120,100),1,0
60 FOR T=0 TO 1000:NEXT T
70 COLOR SPRITE$(0)=CHR$(&H81)+CHR$(&H81
)
80 FOR T=0 TO 1000:NEXT T
90 PUT SPRITE 1,(120,96),14,1
100 GOTO 100
```

行70で，スプライト面0に表示されたスプライトパターン（パターン番号0）の，1ライン目と2ライン目を左に32ドット分ずらしています。



スプライトパターン定義のテクニック

スプライトパターン定義の基本方法は，SPRITE$変数にパターンのデータをCHR$関数を使って代入する方法です。

次のプログラム例では，スプライトパターンのデータをDATA文に書いておき，それをREAD文で読みながらSPRITE$変数に代入していきますが，DATA文中では定義したいスプライトパターンを ．（ピリオド）とO（大文字のオー）により，視覚的に分かるように書くことができるようにしています。

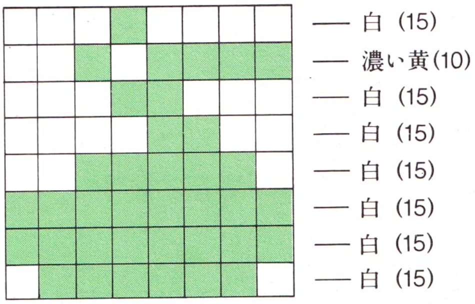
```
10 SCREEN 5,1
20 SP$="":SC$=""
30 FOR SI=0 TO 7
40 READ SQ$,SC:SP=0
50 FOR SJ=1 TO 8
60 SP=SP*2-(MID$(SQ$,SJ,1)="O")
70 NEXT SJ
80 SP$=SP$+CHR$(SP):SC$=SC$+CHR$(SC)
90 NEXT SI
100 SPRITE$(0)=SP$
110 COLOR SPRITE$(0)=SC$
120 PUT SPRITE 0,(120,90),,0
130 GOTO 130
140 '
150 DATA ...O....,15
160 DATA ..O.OOOO,10
170 DATA ...OO...,15
180 DATA ....OO..,15
190 DATA ..OOOOO.,15
200 DATA OOOOOOOO,15
210 DATA OOOOOOOO,15
220 DATA .OOOOOO.,15
```

行150～220のDATA文のデータ部分をみると，定義したいスプライトパターンが．（ピリオド）とOで表わされています。．（ピリオド）は塗りつぶさない部分，Oは塗りつぶす部分です。また，各DATA文の最後の数値データ（行150の場合15）は，そのラインの色を指定するカラーコードです。

DATA文のデータをSPRITE$変数に代入し，COLOR SPRITE$文でカラーの設定をしているのが行20～110の部分です。

ここでは，関数（第8章で説明）を使ってデータを16進数に変換し（行60），最終的に行100，110によりパターンとカラーを定義しています。

この例で定義されるのは次のようなスプライトパターンです。



このプログラムのスプライト定義方法は，ちょっと高度なテクニックですので，これをこのまま公式のように使っても良いでしょう。プログラムとしては長くなりますが，パターンをプログラムリスト上で視覚的に確認できるという長所があり，パターンを作るときもいちいち16進数のデータを作る必要もありません。また，パターンの修正も，DATA文中の．（ピリオド）とOを変更するだけで簡単にできます。

サンプルプログラム

次のプログラムは，スプライトの定義にDATA文を使うテクニックを利用したものです。
この本では説明のされない文(DEFINT文，DEFFN文)もでてきますが，入力して実行してみてください。

このプログラムを実行すると，水鳥の親子づれが池の上を泳いでいきます。

```
10 SCREEN 5,1
20 DEFINT A-Z:X=0:Y=0:Z=0
30 DEFFNX=(Z+240)MOD 256
40 ' *** sprite definition ***
50 RESTORE 380:SN=0:GOSUB 260
60 RESTORE 470:SN=1:GOSUB 260
70 RESTORE 470:SN=2:GOSUB 260
80 RESTORE 470:SN=3:GOSUB 260
90 RESTORE 470:SN=4:GOSUB 260
100 RESTORE 470:SN=5:GOSUB 260
```

```
110 ' *** draw pond ***
120 LINE (0,116)-(255,116),7
130 PAINT (0,118),7
140 ' *** move sprite ***
150 Y=100
160 FOR X=0 TO 255
170  Z=X   :PUT SPRITE 0,(Z,Y),,0
180  Z=FNX:PUT SPRITE 1,(Z,Y),,1
190  Z=FNX:PUT SPRITE 2,(Z,Y),,2
200  Z=FNX:PUT SPRITE 3,(Z,Y),,3
210  Z=FNX:PUT SPRITE 4,(Z,Y),,4
220  Z=FNX:PUT SPRITE 5,(Z,Y),,5
230 NEXT X
240 GOTO 160
250 ' *** sprite definition subroutine *
**
260 SP$="":SC$=""
270 FOR SI=0 TO 7
280 READ SQ$,SC:SP=0
290  FOR SJ=1 TO 8
300   SP=SP*2-(MID$(SQ$,SJ,1)="O")
310  NEXT SJ
320  SP$=SP$+CHR$(SP):SC$=SC$+CHR$(SC)
330 NEXT SI
340 SPRITE$(SN)=SP$
350 COLOR SPRITE$(SN)=SC$
360 RETURN
370 ' *** sprite data ***
380 DATA ...O....,15
390 DATA ..O.OOOO,10
400 DATA ...OO...,15
410 DATA ....OO..,15
420 DATA ..OOOOO.,15
430 DATA OOOOOOOO,15
440 DATA OOOOOOOO,15
450 DATA .OOOOOO.,15
460 ' *** sprite data ***
470 DATA ........,0
480 DATA ........,0
490 DATA ....O...,10
500 DATA ...O.OOO,8
510 DATA ....OO..,10
520 DATA .OO..OO.,10
530 DATA .OOOOOO.,10
540 DATA ..OOOO..,10
```

第6章
日本語を使おう

日本語入力モードにする

- 日本語入力モードとは
- 日本語入力モードに切り換える——CALL KANJI
- 英数字入力モードに戻す——CALL ANK

日本語入力モードとは

MSX_{2+}-BASICでは，漢字を含む文字列を画面に表示したり，プリンターで印字したりすることができます。また，漢字を含む文字列をデータとして扱うことができます。

漢字を扱うことのできる状態を，日本語入力モードと呼ぶことにします。

これに対して，アルファベットや数字，ひらがな，カタカナだけが使える状態を英数字入力モードと呼ぶことにします。

文字コードについて

BASICで色を表わすとき，カラーコードというものを使いました。それと同じように，文字にもそれぞれにコードが割り当てられています。

例えば，大文字のAのコードは10進数で65，16進数で41Hです。(16進数の数値には，10進数の数値と区別するために最後にHをつけることにします。)

また，「漢」という字の文字コードは16進数で3441Hです。「A」などのように，コードが2けたの16進数の文字を1バイトコード文字，「漢」などのように，コードが4けたの16進数の文字を2バイトコード文字と呼びます。

1バイトコード文字には，アルファベットや数字，記号およびひらがな，カタカナがあります。これに対して，2バイトコード文字の中心は漢字です。ただし，2バイトコードの文字の中にも，アルファベットや数字，記号，ひらがな，カタカナが含まれます。例えば，2バイトコードの2341Hは大文字のAを表わします。

MSX-BASICで文字を扱うとき，ふだんはいちいちその文字コードが何かを気にする必要はありませんが，まったく異なる2つの文字グループがある，ということだけ覚えておいてください。(なお，2バイトコードにもJISコード，シフトJISコードなどのコード体系があります。)

日本語入力モードと文字コード

日本語入力モードでは，1バイトコードの文字と2バイトコードの文字を自由に扱えるようになります。どちらのグループの文字を扱うのかは，簡単なキー操作で切り換えます。

日本語入力モードにするには　CALL KANJI

MSX_{2+}-BASICがスタートしたときには，英数字入力モードになっています。このままでは，漢字を表示したり印字したりするプログラムを作ることはできません。そこで，CALL KANJI文を直接モードで実行して，日本語入力モードに切り換えます。

CALL KANJI［漢字フォント］

漢字フォントは，0から3までのいずれかで，その指定のしかたによって次のように設定されます。

漢字フォント	設定
0（または省略）	横16ドット×たて16ドット（全角）
1	MSXの標準漢字ROMを使用。 横12ドット×たて16ドット（全角）に圧縮。
2	漢字フォント0（または省略）のときと同じ。 インターレースモードにより，たて方向が圧縮され，表示行数が増える。
3	漢字フォント1のときのと同じ。 インターレースモードにより，たて方向が圧縮され，表示行数が増える。

では，CALL KANJI文を実行してみてください。

```
CALL KANJI
```

画面はいったんクリアされ，画面の上に「Ok」の表示が現われます。なんとなく文字の大きさが大きくなっていますね。これは，漢字を含む文字を16ドット×16ドットで表示するためです。インターレースモードにより，8ドット×8ドットと同じサ

イズの中に漢字を表示させたければ,

```
CALL KANJI2
```

を実行します。

CALL KANJI文を実行する前にプログラムを入力したり,作成途中でGOTO文などを使って実験をしていた場合は,必ずいったんプログラムをディスクにセーブしてから日本語入力モードに切り換えてください。日本語入力モードに切り換えると,コンピューターのメモリー構成が日本語処理用に変更されてしまい,プログラムの実験などを継続できなくなります。

なお,漢字を表示させるコマンドに,もうひとつPUT KANJI文がありますが,PUT KANJI文を使って漢字を含むプログラムを書くことはできません。

英数字入力モードに戻すには　CALL ANK

もとの英数字入力モードに戻すには,CALL ANK文を直接モードで実行します。

コール アンク
CALL ANK

日本語を使ってプログラムを作る

●連文節変換による入力方法

こんなプログラムを作ってみよう

では，日本語を使ったプログラムを作ってみることにしましょう。ここでは，どうやって日本語を入力するかの練習をしてみますので，プログラムそのものは次のような簡単なものを例にしてみます。

```
10 CALL KANJI
20 LOCATE 3,5
30 PRINT "MSXで漢字を使おう"
```

行10のCALL KANJI文は，英数字入力モードでこのプログラムを実行しても漢字が表示できる日本語入力モードに切り換わるとともに，画面がクリアされるようにするために使われています。(なお，日本語入力モードで画面をクリアするときには，CALL CLSというコマンドを使います。)

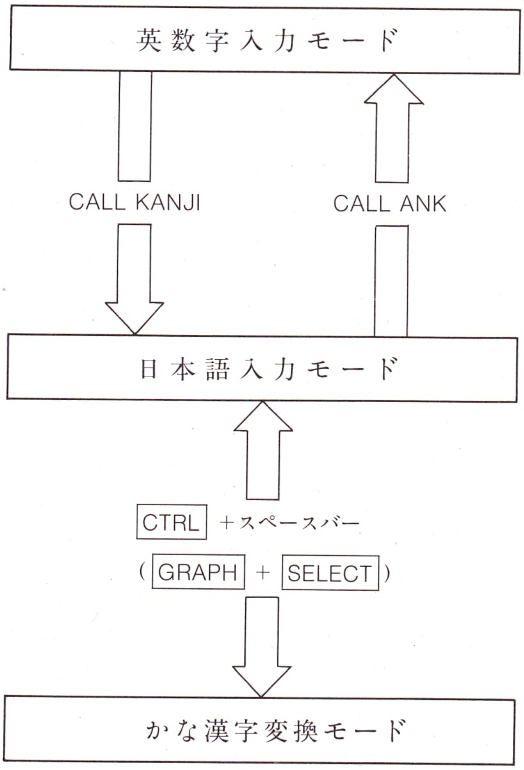
かな漢字変換モードへの切り換え（CTRL＋スペースバーまたはGRAPH＋SELECT）

漢字を表示させるには，まずひらがなをキーボードから入力して，それを漢字に変換する，という作業を行ないます。

ところで，CALL KANJI文で日本語入力モードに切り換えても，すぐにかなから漢字への変換ができるわけではありません。日本語入力モードのときに，CTRLキーを押しながらスペースバーを押すことによって，かな漢字変換ができるようになります。(あるいは，GRAPHキーを押しながらSELECTキーを押します。)

もう一度CTRL＋スペースバー（またはGRAPH＋SELECT）を押すと，かな漢字変換モードは解除されます。

以上をまとめると，次のようになります。



日本語とプログラム

プログラム中，かな漢字変換による日本語の文字列が使えるのは，PRINT文で表示させたい文やデータなどに限ります。プログラムそのもの，つまり行番号や命令語(CALL KANJIやPRINTなど)，プログラム上の記号("や：など)をかな漢字変換の方法で入力すると，プログラムにはならなくなります。

なお，かな漢字変換の方法でも，プログラムをすべて半角文字で入力すればプログラムとして動作します。いいかえれば，プログラムそのものに全角文字が入っていてはいけないということです。

連文節変換によりプログラムを入力する

ご注意

MSX$_{2+}$-BASICの標準仕様では，かな漢字変換の方法としては単漢字変換という方法だけが使えます。この方法では，表示したい漢字を一字ずつ，その漢字の音読みをひらがなで入力しては漢字に変換していきます。

MSXの拡張仕様であるMSX-JEが内蔵されていないMSX$_{2+}$のコンピューターに，MSX-JE

のカートリッジを取り付けていない場合は，連文節変換でかなを漢字に変換することができません。単漢字変換の方法については，お使いになる機器の説明書をお読みください。

連文節変換のルール

MSX-JE内蔵のコンピューターや，MSX-JEのカートリッジ（ソニーHBI-J1など）を取り付けたコンピューターでは，連文節変換でかなを漢字に変換することができます。

連文節変換では，まとまった文章をひらがなで入力し，それを文節ごとに漢字とかなに変換していきます。文節とは，それ以上区切ると，それだけでは意味がなくなるものが出てくるような，文の単位です。

たとえば，「漢字を使おう」は，「漢字を」と「使おう」の2つの文節でできています。

入力文字の切り換え

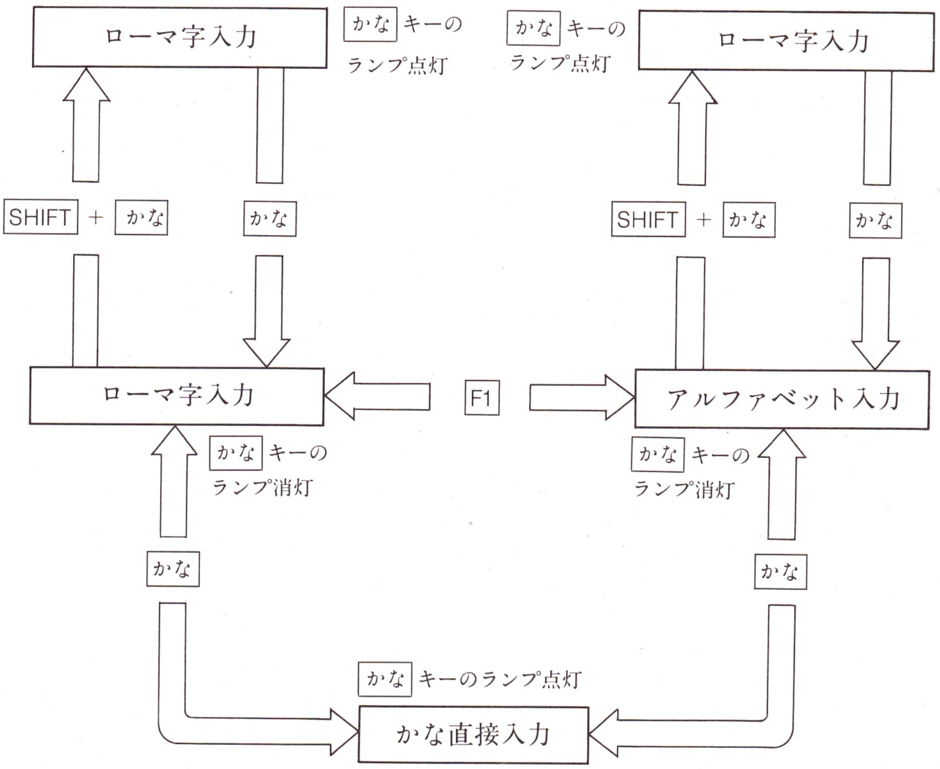
かな漢字変換モードにしたとき，最初はキーを押してもかなに変換されず，アルファベットのまま入力される状態（アルファベット入力）になっている場合があります。

この場合，かなを，ローマ字のつづりで入力したいときは，[F1]キーを押します。（ローマ字入力）

キーボードのかな表記にしたがってかなを入力したいときは，[かな]キーを押して，[かな]キーのランプを点けます。（かな直接入力）

なお，アルファベット入力またはローマ字入力の状態（[かな]キーのランプが消えている状態）で，[SHIFT]キーを押しながら[かな]キーを押して[かな]キーのランプを点けると，ローマ字入力の状態になります。

これら3種類の入力モードの切り換えかたをまとめると，つぎのようになります。



プログラムの入力

例題のプログラムを入力しながら，変換の方法を練習しましょう。

1　CALL KANJI文を直接モードで実行して，日本語入力モードにします。

2　行30の，「MSXで漢字を使おう」の直前までを入力します。

```
10 CALL KANJI
20 LOCATE 3,5
30 PRINT "
```

3　CTRLキーを押しながらスペースバーを押して，(あるいはGRAPHキーを押しながらSELECTキーを押して)，かな漢字変換の準備をします。画面のいちばん下の行が，かな漢字変換に使われる部分です。

4　かなキーのランプが消えていることを確かめます。(点いていたら，かなキーを一回押します。)

5　「MSX」を入力します。

画面の下に，「MSX」が表示されます。

```
ＭＳＸ
```

6　RETURN キーを押します。

「MSX」がプログラムに追加されます。

```
10 CALL KANJI
20 LOCATE 3,5
30 PRINT "ＭＳＸ
```

このとき，日本語入力モードになっているので，「MSX」は，全角の文字で入力されます。(漢字を含む日本語のグループに属する英字とお考えください。)

7　かな入力モードにします。

8　かなで「で」を入力します。

かな直接入力の場合，て ゛と押します。

ローマ字入力の場合，D Eと押します。

画面の下に，背景が白で「で」と表示されます。

```
で
```

9　「で」は漢字に変換する必要がないので，RETURN キーを押します。

プログラムに「で」が追加され，画面下の「で」は消えます。

```
10 CALL KANJI
20 LOCATE 3,5
30 PRINT "ＭＳＸで
```

10　かなで「かんじをつかおう」と入力します。

画面の下に，「かんじをつかおう」と表示されます。

```
かんじをつかおう
```

11　スペースバーを押します。

最初の文節である，「かんじを」の部分が漢字に変換されます。

```
漢字を使おう
```

変換された部分の背景は明るい青になっています。

12　正しく変換されているので，決定するために RETURN キーを押します。

「漢字を」がプログラムに追加されます。

```
10 CALL KANJI
20 LOCATE 3,5
30 PRINT "ＭＳＸで漢字を
```

画面の下には，「使おう」が残ります。

13　これも正しく変換されているので，RETURNキーを押します。

```
10 CALL KANJI
20 LOCATE 3,5
30 PRINT "ＭＳＸで漢字を使おう
```

14　CTRLキーを押しながらスペースバーを押して，（あるいはGRAPHキーを押しながらSELECTキーを押して）かな漢字変換モードを解除します。

15　行30の最後の　"　を入力して，プログラムの完成です。

```
10 CALL KANJI
20 LOCATE 3,5
30 PRINT "ＭＳＸで漢字を使おう"
```

変換作業に使うキー

上の例では，スペースバーを一回押しただけで目的の漢字に変換されました。一回で変換されないときは，希望の漢字に変換されるまでスペースバーを押します。この操作も含めて，変換作業に使うキーをまとめておきましょう。

表の中で，「確定」ということばが使われていますが，これは，表示されている漢字に決定することで，確定した文字はもう変換することはできません。

働き	キー	意味	
		変換前	変換中
変換	スペースバー	変換開始	次の候補に進める。
	GRAPHキー	—	前の候補に戻す。
	△キー	変換開始	変換対象を1ブロック長くする。
	▽キー	変換開始	変換対象を1ブロック短くする。

働き	キー	意味	
		変換前	変換中
確定	RETURNキー	ひらがなのままで確定する。	変換中の部分を確定する。
全確定	F5キー	入力中の文字を，現在表示されている状態で全部確定する。	
カタカナ変換	F2キー	入力中の文字を全角のカタカナに変換する。(漢字もカタカナになる。)	
半角変換	F3キー	入力中の文字を半角に変換する。(ひらがなと漢字は半角のカタカナになる。)	
コード変換	F4キー	入力されたJISコードを，文字に変換する。	
カーソル移動／変換中断	◁　▷キー	カーソルを1字分左右に移動する。	すべてひらがなに戻す。
	SHIFT＋◁ SHIFT＋▷	カーソルを左端または右端に移動する。	すべてひらがなに戻す。
後退削除	BSキー	カーソルの1字前の文字を削除。	—
削除	DELキー	カーソルの位置の文字を削除。	—
全角スペース	SHIFT＋スペースバー	全角のスペースを入力する。	次の候補に進める。
モード切り換え	F1キー	アルファベット入力とローマ字入力を切り換える。	
作業中断／変換中断	ESCキー	すべての文字を消す。	すべてひらがなに戻す。
	CTRL ＋ STOPキー	すべての文字を消す。	

上の表と一部内容が重なりますが，連文節変換でのいろいろな文字の入力方法をまとめておきます。

文字	入力のしかた	
	日本語入力モードで	かな漢字変換モードで
ひらがな（全角）	かなキーのランプが点いている状態で入力。注1)	
全角のカタカナ	—	ひらがなを入力したら，F2キーで変換。
半角のカタカナ	かなキーのランプが点いている状態でCAPSキーを押してCAPSキーのランプを点ける。	ひらがなを入力したら，F3キーで変換。
全角の記号	—	JISコードを入力してF4キーで変換。
半角の記号	アルファベット入力の状態で入力できる記号だけ入力できる。　注2)	—
漢字	—	読みをひらがなで入力して連文節変換で変換。

注1) ローマ字入力のときはかなキーのランプは消えている場合もあります。

注2) 日本語入力モードでは，GRAPHキーによる記号類の入力はできません。

画面をクリアするには

CALL KANJI文で日本語入力モードにしてあるときは，画面をクリアするのにCLS文は使えません。CALL CLS文を使ってください。また，カラーパレットを設定するときもCOLOR文ではなくCALL PALETTE文を使います。

第7章
音楽を演奏しよう

メロディーの演奏

- MSXでメロディーを演奏するには——PLAY
- 音の高さを決める——A～Gサブコマンド，Oサブコマンド
- 音の長さを決める・休符を入れる——Lサブコマンド，Rサブコマンド
- 演奏の速さを決める——Tサブコマンド
- 音色を変える——Sサブコマンド，Mサブコマンド
- 和音を演奏する

MSXと音楽演奏　PLAY

MSXのコンピューターには，PSG（ピーエスジー）（Programmable Sound Generator（プログラマブルサウンドジエネレーター））と呼ばれる，音楽演奏用のLSIが組み込まれています。MSX-BASICには，このPSGを使って音楽を演奏するための命令語が用意されています。

音を出してみる

PSGを使って音を出すための命令語にPLAYがあります。ものはためし，とりあえず音を出してみましょう。次の文を実行してみてください。

```
10 PLAY "CEGEC"
```

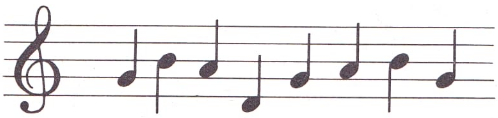
ドミソミド，と聞こえたでしょう。

ところで，このPLAY文，かたちがPRINT文とそっくりですね。PRINT文の場合，画面に表示したい文字を”　　　”で囲みました。PLAY文では，演奏したい楽譜の内容を”　　　”で囲むのです。ただし，楽譜には音符だけではなく，休符や速さを決める記号など，さまざまな要素がふくまれています。PLAY文の”　　　”の中に，楽譜で表現するのと同じ内容を盛り込むために，いろいろなサブコマンドというものを使用します。いま実行したPLAY文の中のCEGECもサブコマンドです。これから，いくつかのサブコマンドを使ってプログラムを作ってみましょう。

音の高さを決める　A～Gサブコマンド、Oサブコマンド

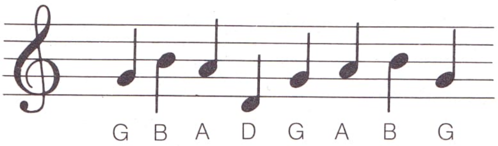
```
10 PLAY "O4"
20 PLAY "GBADGABG"
```

このプログラムを実行すると、次の楽譜が演奏されます。



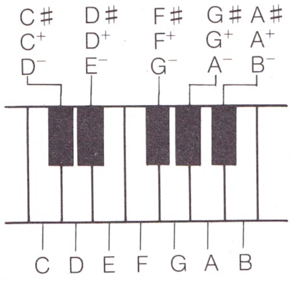
音程を決めるA～Gサブコマンド

いまのところ、何分の何拍子かもはっきりしていませんし、音符はすべて4分音符ですが、それは気にせずに行20をみてください。このPLAY文中のサブコマンドであるGBADGABGが、音符のひとつひとつを表わしているのです。



AからGのサブコマンドを使って、音程を示しているわけです。この楽譜には出てきませんが、#や♭で半音上げたり下げたりする場合、サブコマンドでは#または＋（半音上げる）と－（半音下げる）の記号を使います。

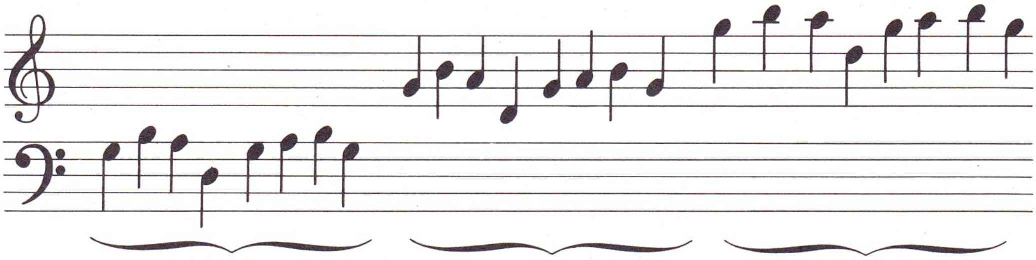
これをピアノの鍵盤でまとめると次のようになります。



オクターブを決めるOサブコマンド

MSX-BASICでは，8オクターブの音域にわたる演奏ができます。行10をみてください。O4というサブコマンドが，4番目のオクターブを示しています。

では，行10をPLAY "O3"，続いてPLAY "O5" に変更してプログラムを実行してみてください。



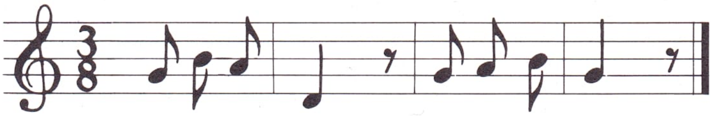
MSX-BASICのPLAY文では，O1からO8までの8つのオクターブを指定することができます。BASICをスタートさせたときは，O4のオクターブに設定されます。もちろん，次のようにサブコマンドの途中でオクターブをかえることもできます。

```
10 PLAY "O4 GBAD O5 GABG"
```

(サブコマンド列の中のスペースはなくてもかまいません。)

音の長さを決める　Lサブコマンド

演奏する楽譜をさらに整えていきます。



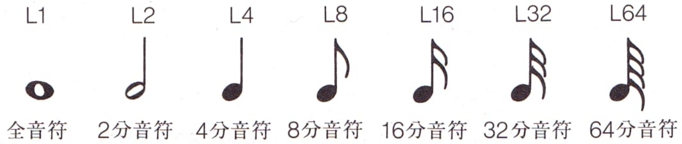
この楽譜を演奏するプログラムは次のようになります。

```
10 PLAY "O4"
20 PLAY "L8 GBA L4 D R8 L8 GAB L4 G R8"
```

行20にいくつかのサブコマンドが追加されました。

長さを決めるLサブコマンド

L1からL64までのサブコマンドで、そのあとに続く音程のサブコマンドの長さを決めます。



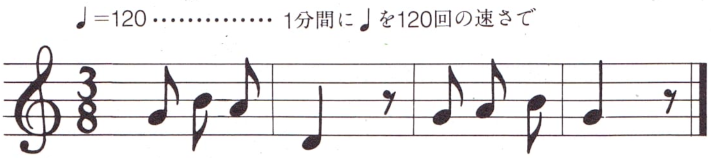
BASICがスタートした状態では、L4（4分音符）の長さが選ばれます。

休符を示すRサブコマンド

休符は，Rサブコマンドを使って表わします。

演奏の速さを決める　Tサブコマンド

BASICがスタートした状態では、1分間に4分音符を120回演奏する速さに設定されています。楽譜では、これを次のように表わします。



MSX-BASICでは、これをTサブコマンドを使ってT120のように書きます。TサブコマンドはT32からT255の範囲で使うことができます。プログラムを次のように変更しましょう。

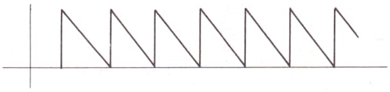
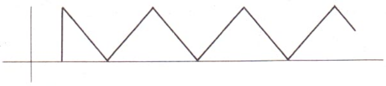
```
10 PLAY "T60 O4"
20 PLAY "L8 GBA L4 D R8 L8 GAB L4 G R8"
```

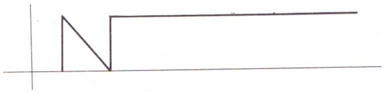
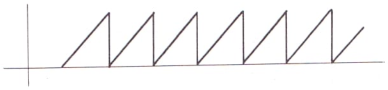
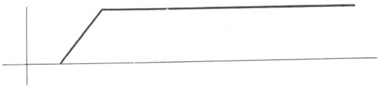
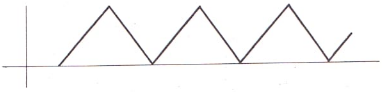
行10に追加されたT60のサブコマンドにより、1分間に4分音符を60回演奏する速さになりました。実行してみると、ずいぶん遅くなったことが分かります。

音色を変える　Sサブコマンド、Mサブコマンド

最後に、SサブコマンドとMサブコマンドを使っておもしろい効果を出してみましょう

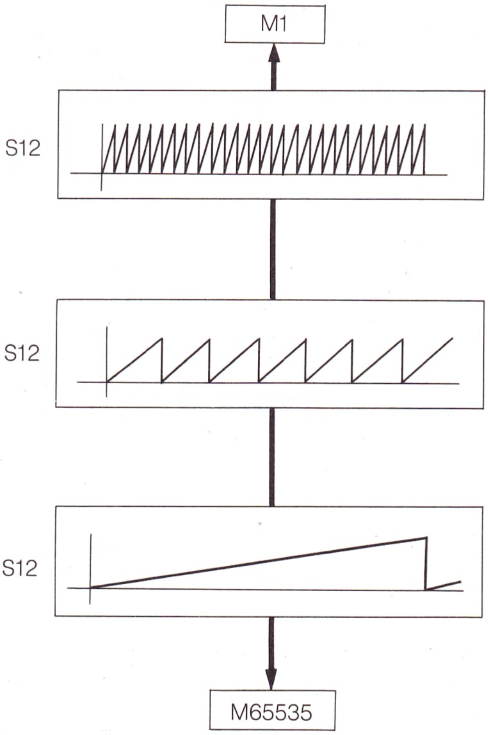
Sサブコマンドは、音量の変化のパターンを決めるサブコマンドです。次の図をみてください。

Sサブコマンド	パターン
S0、S1、S2、S3、S9	
S4、S5、S6、S7、S15	
S8	
S10	

Sサブコマンド	パターン
S11	
S12	
S13	
S14	

初期設定はS1です。

また，Mサブコマンドを使うと，Sサブコマンドで選んだパターンの周期を設定することができます。Mサブコマンドは M1からM65535の範囲で使うことができ，数字が大きいほど周期が大きくなります。たとえば，同じS12サブコマンドでも，Mサブコマンドでパターンの変化の周期をいろいろと変えることができます。(BASICがスタートしたときはM255の周期がえらばれます。)



では、このふたつのサブコマンドを実際に使ってみましょう。

```
10 PLAY "T60 O4"
15 PLAY "S0"
20 PLAY "L8 GBA L4 D R8 L8 GAB L4 G R8"
```

このプラグラムでは、音がプツッ、プツッととぎれてしまい、メロディーになりません。これは、Mサブコマンドを実行していないので自動的にM255の状態に設定されているからです。そこで、次のようにMサブコマンドを追加します。

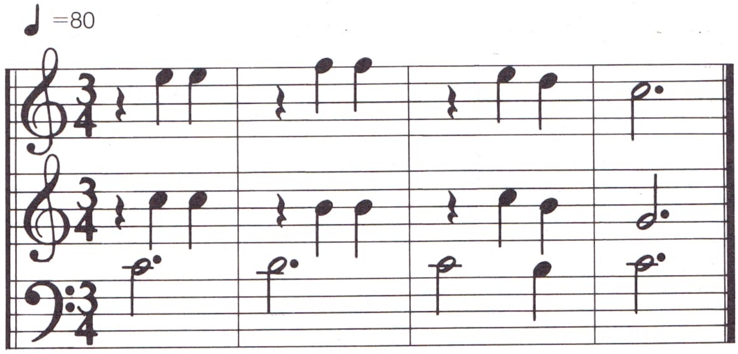
```
10 PLAY "T60 O4"
15 PLAY "S0M20000"
20 PLAY "L8 GBA L4 D R8 L8 GAB L4 G R8"
```

これでチャイムのような音になりました。

和音を演奏する

MSXコンピューターのPSGは、同時に3つのメロディーを演奏することができます。

また、PLAY文のサブコマンドも文字型の変数に入れておくことができます。このふたつを利用して、次の楽譜を演奏するプログラムを作ってみましょう。




```
10 A$="R4 L4 O5 EE R4 FF R4 ED O5 L2 C."
20 B$="R4 L4 O5 CC R4 O4 BB R4 O5 C O4 B L2 G."
30 C$="O4 L2 C.D.C O3 L4 B O4 L2 C."
40 PLAY "T80 S0 M10000 XA$;","T80 S0 M10000 XB$;",
   "T80 S0 M25000 XC$;"
```

楽譜に示された3つのメロディーが、高音部からそれぞれ変数A$、B$、C$に代入されています。

ここで、音程を表わすA～Gサブコマンドに．（ピリオド）のつけられたものがあります。ピリオドは、その音符の長さを1.5倍にのばす働きをします。たとえば行10の最後の"L2C."では、Lサブコマンドで2分音符を指定していますが、ピリオドにより1.5倍、すなわち付点2分音符となります。

行40のPLAY文で，3つのメロディーを同時に演奏しています。演奏したいサブコマンドを，それぞれ"　　　"で囲んで，（カンマ）でつなげると同時に演奏されます。

サブコマンドのなかで，あらかじめ変数に入れておいたものを使うときは，変数の前にX，後ろに;（セミコロン）をつけることになっています。上のプログラムで使いかたを確認してください。

効果音を作る

●PSGとレジスター
●効果音を作る――SOUND

PSGとレジスター

PLAY文を使うと，MSXコンピューターのPSGでメロディーを演奏することができました。このPSGには，16のレジスターが備わっています。レジスターとは，データを一時的に保存しておく場所のことです。PSGは，これらのレジスターに入れられたデータの値によっていろいろな音を出します。

効果音を作る　SOUND

PLAY文では，メロディーを演奏するのにサブコマンドを使いました。コンピューターは，与えられたサブコマンドからデータの値を計算してPSGのレジスターに入れます。その結果，指定どおりの音が出ることになります。

レジスターに入れるデータを直接指定できるSOUND文

MSX-BASICには，PSGを操作する命令語としてPLAY文のほかにSOUND文があります。SOUND文では，サブコマンドのようなものは使わず，16のレジスターのうちの14個にどんなデータを入れるのかを直接指定することによって音を出します。ですから，PSGのそれぞれのレジスターにどんなデータを入れるとどんな音が出るのかが分かれば，SOUND文でいろいろな音を出すことができます。SOUND文は，メロディーの演奏よりも効果音（シュッシュッというSLの音やザーザーという波の音など）を作るのに適しています。

PSGのレジスターの働き

MSXコンピューターのPSGは，同時に3つの音を出すことができます。つまり3つのチャンネルを持っているのです。3つのチャンネルを，それぞれチャンネルA，B，Cと呼びます。以下に，PSGの14のレジスターの働きをまとめておきます。

レジスター番号	働　　き	データの範囲
0	チャンネルAの周波数を決める。	0～255
1		0～15
2	チャンネルBの周波数を決める。	0～255
3		0～15
4	チャンネルCの周波数を決める。	0～255
5		0～15
6	ノイズの周波数を決める。	0～31
7	どのチャンネルから音やノイズを出すかを決める。	0～63
8	チャンネルAの音量を決める。	0～15 16にすると音量を変化させることができる。
9	チャンネルBの音量を決める。	
10	チャンネルCの音量を決める。	
11	音量変化パターンの周波数を決める。	0～255
12		0～255
13	音量変化のパターンを決める。	0～14

SOUND文でレジスターにデータを入れるには、SOUND文を次のように使います。

SOUND（サウンド）　レジスター番号，データ

自動車とSLを走らせてみる

それではSOUND文を使って効果音を出してみましょう。ここでは、ブーンという自動車のエンジン音とシュッシュッというSLの音を同時に出してみます。そのために、どのチャンネルからどんな音を出すのか、そのためにはどのレジスターにどんなデー

タを入れるのかを決めていかなくてはなりません。

各チャンネルの周波数を決める

まず，ブーンというエンジン音ですが，これはチャンネルAから低い周波数の音を一定の音量で出し続けることにします。周波数とデータの値の関係は次の式で表わされます。

$$\frac{1789772.5\ (\mathrm{Hz})}{16\times 出力したい周波数\ (\mathrm{Hz})}=256\times（レジスター1，3，5のデータ）$$

$$+（レジスター0，2，4のデータ）$$

チャンネルAの周波数はレジスター0とレジスター1の値によって決まります。ここでは，約30Hzにすることにします。すると，

$$\frac{1789772.5}{16\times 30}=3729\fallingdotseq 256\times 14+145$$

という式が成り立ちますから，レジスター0には145，レジスター1には14を入れれば良いことになります。

チャンネルBとCからは音は出さないことにしますから，レジスター2，4とレジスター3，5にはとくにデータを入れる必要はありません。

ノイズの周波数を決める

SLのシュッシュッという音は，ノイズ音を使って作ることにします。ノイズ音はサーッ，ザーッという音ですが，この音に音量変化のパターンをあてはめてシュッシュッと聞こえるようにしてみます。ここではとりあえず基本となるノイズの周波数を決めます。

ノイズの周波数はレジスター6に入れるデータの値によって決まります。データと周波数の間には次の関係があります。

$$データの値=\frac{1789772.5\ (\mathrm{Hz})}{16\times ノイズ周波数\ (\mathrm{Hz})}$$

ノイズの周波数は約7500Hzということにしましょう。すると，

$$\frac{1789772.5}{16\times 7500}\fallingdotseq 15$$

で，レジスター6には15を入れれば良いということになります。

音を出すチャンネルを決める

レジスター7に入れるデータの値によって，どのチャンネルから音を出すかを決めます。

	ノイズ			音		
チャンネル	C	B	A	C	B	A
	32	16	8	4	2	1

上の表をみて，使うチャンネルに対応する数値を合計して63から引くと，レジスター7に入れるデータの値が求められます。

ここでは，チャンネルAから周波数50Hzの音を，チャンネルBからノイズを出すことにします。すると，

$$63-(16+1)=46$$

で，レジスター7には46を入れることになります。

音量変化のパターンと周期を決める

PLAY文のSサブコマンドで選ぶことのできたパターンが，SOUND文でも使えます。使いたいパターン番号をレジスター13に入れます。パターンの番号については148ページをみてください。

ここでは，シュッシュッという音を出すために，パターン12を使うことにしましょう。これで，レジスター13に入れるデータは12に決まりました。

次に，PLAY文のMサブコマンドでパターンの周期を決めたように，SOUND文でも周期を決めなければなりません。パターン周期はレジスター11と12に入れるデータで決まり，次の式が成り立ちます。

$$\frac{1789772.5(\text{Hz})}{256\times\text{周期}(\text{Hz})}=256\times(\text{レジスター12のデータ})+(\text{レジスター11のデータ})$$

ここでは周期を約5.5Hzに設定することにします。すると，

$$\frac{1789772.5}{256\times5.5}\fallingdotseq1271=256\times4+247$$

となり，レジスター11に247，レジスター12に4を入れることになります。

各チャンネルの音量を決める

チャンネルA，B，Cの音量は，それぞれレジスター8，9，10に入れるデータの値できまります。データは0から15まで，および16のいずれかで，0から15までの場合15を最大として音量を調節できます。16にするとレジスター11から13までのデータにより音量が変化するようになります。

チャンネルAからは一定の音を出しますから，レジスター8には0から15までのどれかを入れることになります。ここでは15を入れることにしましょう。
チャンネルBからは音量の変化するノイズを出しますから，レジスター9には16を入れます。
チャンネルCは使いません。したがってレジスター10には何も入れません。

プログラムを作る

これで，各レジスターに入れるデータがすべて決まりました。
SOUND文を使ってプログラムにしてみましょう。

```
10 SOUND 0,145
20 SOUND 1,14
30 SOUND 6,15
40 SOUND 7,46
50 SOUND 13,12
60 SOUND 11,247
70 SOUND 12,4
80 SOUND 8,15
90 SOUND 9,16
```

プログラムを実行すると，ブーンという音とシュッシュッという音が聞こえます。
自動車を運転していると，その横をSLが並んで走っていく，そんなふうに聞こえますか。
音を止めるには CTRL + STOP を押してください。

FM音源を使う

●MSX-MUSICの拡張BASICとFM音源
●使うチャンネルを決める——CALL MUSIC
●音色を決める——CALL VOICE
●演奏する——PLAY

FM音源（MSX-MUSIC）について

FM音源（MSX-MUSICと呼ばれます。以下の説明ではFM音源と表わします）を内蔵したMSXコンピューターで，FM音源を使った音楽演奏ができます。FM音源を内蔵していないMSXコンピューターでは，同等の機能を持ったカートリッジを使用すれば，以下に説明する命令語（拡張コマンド）を使えるようになります。

使うチャンネルを決める　CALL MUSIC

FM音源には9つのチャンネルがあります。FM音源を使って演奏するときはPLAY文を使います。拡張BASICでは，FM音源を使用する場合ひとつのPLAY文で最大12のサブコマンド列を使うことができます。（”　　　”で囲まれたサブコマンドをひとつの列と数えます。）
FM音源を使って演奏するときは，まずPLAY文の各サブコマンド列をどのチャンネルから演奏するかを決めて，それを指定する必要があります。そのために，拡張コマンドであるCALL MUSIC文を実行します。

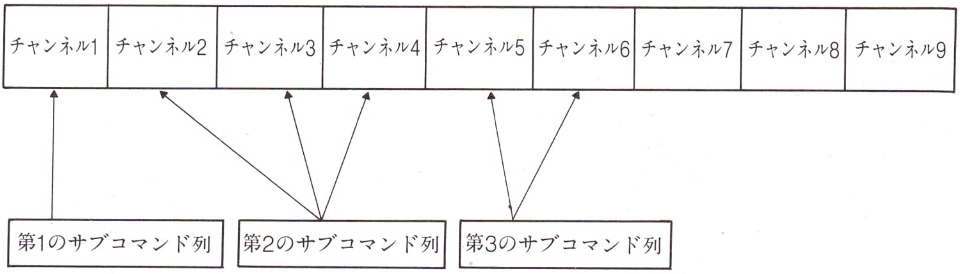
CALL MUSIC（コール ミユージツク）［(［モード］［,［0］［,PLAY文第1サブコマンド列用のチャンネル数［,第2サブコマンド列用のチャンネル数［,...,第9サブコマンド列用のチャンネル数]]]]]]]]]]])］

なお，CALL MUSIC文を実行しないとFM音源を使う命令語を使えませんので，FM音源演奏プログラムを作る場合，必ず最初にCALL MUSIC文を実行するようにしてください。

PLAY文のサブコマンド列とチャンネル

PLAY文のサブコマンド列は，第1サブコマンド列から始めて，順にFM音源のチャンネル1から割り当てていきます。このとき，FM音源のチャンネルはぜんぶで9つであるということに注意します。

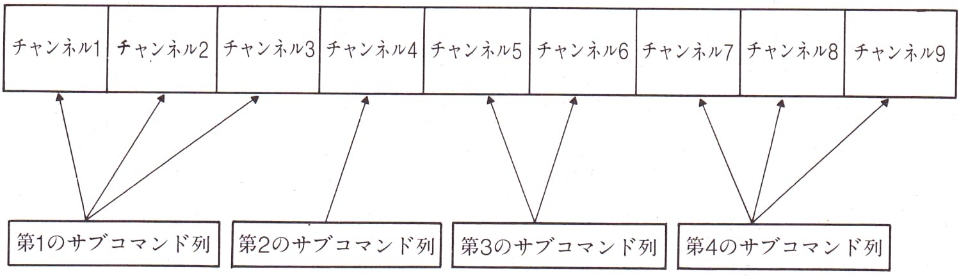
たとえば，次のような割り当てができます。



この割り当てをするには，

```
CALL MUSIC(0,0,1,3,2)
```

を実行します。この場合，まだ3つのチャンネルがあまっています。



この割り当てをするには，

```
CALL MUSIC(0,0,3,1,2,3)
```

を実行します。この場合は，すべてのチャンネルを使っているので，PLAY文ではFM音源用のサブコマンド列は4つしか使えません。

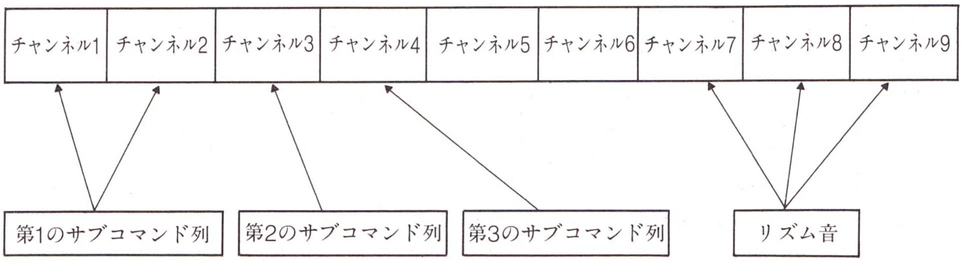
リズム音を使う

リズム音とは，ドラムやシンバルなどのリズム楽器を模した音のことです。FM音源使用のPLAY文では，リズム音もサブコマンド列としてひとつ指定することができます。

リズム音を出すときは，CALL MUSIC文のモードを1にします。

リズム音とチャンネル

リズム音は，FM音源のチャンネル7，8，9から出力されます。したがって，リズム音を使用する場合はチャンネル1から6までをメロディー演奏に使うことができます。たとえば次のようなチャンネル割り当てができます。



この割り当てをするには，

```
CALL MUSIC(1,0,2,1,1)
```

を実行します。

各チャンネルの音色を決める　CALL VOICE

MSXのFM音源には，あらかじめ次の表の64種類の音色が組み込まれています。それぞれの音色には，@0から@63までの番号がついています。

音色番号	音色	音色番号	音色
@0	ピアノ1*	@32	ピアノ3
@1	ピアノ2	@33	エレクトリックピアノ2*
@2	バイオリン*	@34	サンツール2
@3	フルート1*	@35	ブラス
@4	クラリネット*	@36	フルート2
@5	オーボエ*	@37	クラビコード2
@6	トランペット*	@38	クラビコード3
@7	パイプオルガン1	@39	琴2
@8	シロフォン（木琴）	@40	パイプオルガン2
@9	オルガン*	@41	PohdsPLA
@10	ギター*	@42	PohdsPRA
@11	サンツール1	@43	チャーチオルガンL
@12	エレクトリックピアノ1*	@44	チャーチオルガンR
@13	クラビコード1	@45	シンセ・バイオリン
@14	ハープシコード1*	@46	シンセ・オルガン
@15	ハープシコード2	@47	シンセ・ブラス
@16	ビブラロン*	@48	ホルン*
@17	琴1	@49	三味線
@18	太鼓	@50	マジカル

音色番号	音色	音色番号	音色
@19	エンジン1	@51	フワワ
@20	UFO	@52	ワンダーフラット
@21	シンセサイザーベル	@53	ハードロック
@22	チャイム	@54	マシン
@23	シンセ・ベース*	@55	マシンV
@24	シンセサイザー*	@56	コミック
@25	シンセ・パーカッション	@57	SE-コミック
@26	シンセ・リズム	@58	SE-レーザー
@27	ハーモ・ドラム	@59	SE-ノイズ
@28	カウベル	@60	SE-星1
@29	ハイハット	@61	SE-星2
@30	スネア・ドラム	@62	エンジン2
@31	バス・ドラム	@63	無音

以上の音色から，希望の音色を選んで，FM音源の各チャンネルに割り当てることができます。そのためには，拡張コマンドであるCALL VOICE文を使います。

CALL VOICE（[チャンネル1の音色]，[チャンネル2の音色]，．．．，[チャンネル9の音色]）

なお，上の表で＊記号のついていない音色どうしは同時に2つ以上設定することはできません。そのような設定をすると，CALL VOICE文で最後のチャンネル番号に設定した音色ひとつだけが有効になります。（同時に指定した＊のついた音色は，すべて有効になります。）
たとえば，チャンネル1をピアノ1に，チャンネル2をバイオリンに，チャンネル3をクラリネットに設定するときは，

```
CALL VOICE(@0,@2,@4)
```

を実行します。

FM音源で演奏する　PLAY

FM音源を使った演奏の場合もやはりPLAY文を使いますが，PLAY文の構造が次のようになります。

PLAY [#モード，] サブコマンド1 [，サブコマンド2]．．．[，サブコマンド12]

ここで，モードを2または3にするとFM音源を使っての演奏ができます。
モードを0にするか，あるいは省略すると，PSGだけを使っての演奏となります。したがって，144ページからのPLAY文の説明と同じことになります。このときはサブコマンド列は3つまでしか使えません。
リズム音をつかう場合，リズム音のサブコマンド列はメロディーの最後のサブコマンド列の次に続けます。さらにPSGからも音を出したいときは，PSG用のサブコマンド列をその後に続けます。リズム音を使用しない設定をした場合，リズム音用のサブコマンド列の部分を，（カンマ）とともに省略しておかなくてはなりません。したがって，サブコマンド列の書きかたは，

●CALL MUSICのモードが0の場合
　FM音源用サブコマンド列（9つまで），PSG用サブコマンド列（3つまで）

●CALL MUSICのモードが1の場合

FM音源用サブコマンド列(6つまで)，リズム音サブコマンド列，PSG用サブコマンド列（3つまで）

となります。

リズム音サブコマンドの書きかた

リズム音のサブコマンドには次のものがあります。

サブコマンド	条件	働き
B	—	バスドラム音を出す。
S	—	スネアドラム音を出す。
M	—	タムタム音を出す。
C	—	シンバル音を出す。
H	—	ハイハット音を出す。
!	—	直前の楽器にアクセントをつける。
n	1≦n≦64	直前までの楽器音を出し，n分音符分待つ。
Vn	0≦n≦15	アクセントのついていない音の音量を設定する。(省略するとV8)
@An	0≦n≦15	アクセントつきの音の音量を設定する。
Tn	32≦n≦255	テンポを決める。(省略するとT120)
@Vn	0≦n≦127	音量を細かく設定する。
Rn	1≦n≦64	休符を指定する。(初期値はR4)

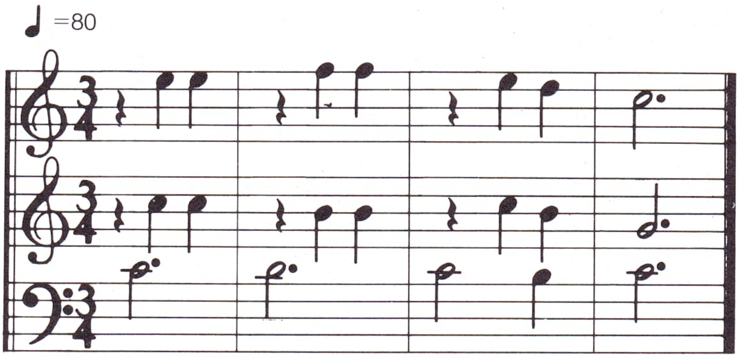
たとえば，次のように，サブコマンドを書きます。

```
"BSH4H4S!4H4"
```

このサブコマンドでは，バスドラム，スネアドラム，ハイハットを鳴らして4分音符分待ち，ハイハットを鳴らして4分音符分待ち，スネアドラムをアクセントつきで鳴らして４分音符分待ち，ハイハットを鳴らして4分音符分待ちます。

プログラムを作る

FM音源を使った演奏のプログラムをひとつ作ってみましょう。
演奏するのは，151ページの練習でも使った次の楽譜です。



ここでは，高音部の和音をピアノで，低音部をオーボエで演奏することにします。

```
10 CALL MUSIC(0,0,1,1,1)
20 CALL VOICE(@0,@0,@5)
30 A$="R4 L4 O5 EE R4 FF R4 ED O5 L22C."
40 B$="R4 L4 O5 CC R4 O4 BB R4 O5 C O4 B L2 G."
50 C$="O4 L2 C.D.C O3 L4 B O4 L2 C."
60 PLAY #2,"T80XA$;","T80XB$;","T80XC$;"
```

第8章
関数に強くなろう

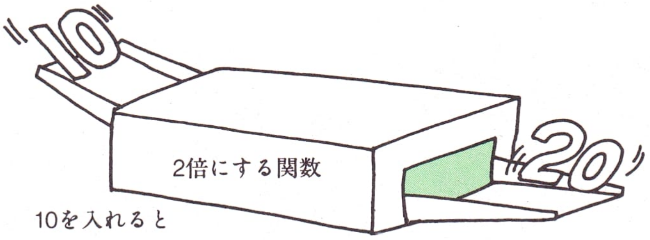
数値型の関数

- 関数とは
- いろいろな数値型関数
- ランダム関数

関数とは

BASICのことばは，大きく命令語と関数のふたつに分けることができます。
今まで「入門編」,「応用編」をとおしてこのうちの命令語について勉強してきました。
この章では，関数というものに焦点をあてて話をすすめたいと思います。

BASICの関数とは，基本的にはある値を加工して，その結果を出してくれる箱のようなものと考えてください。
たとえば，「数を2倍にする」という働きを持つ関数があるとします。(このような関数はMSX_{2+}-BASICにはありません。）すると，この関数に10という値を入れると20になって出てきます。100を入れれば200に，450を入れれば900に，と入れた数はすべて倍になって出てきます。



数値型の関数

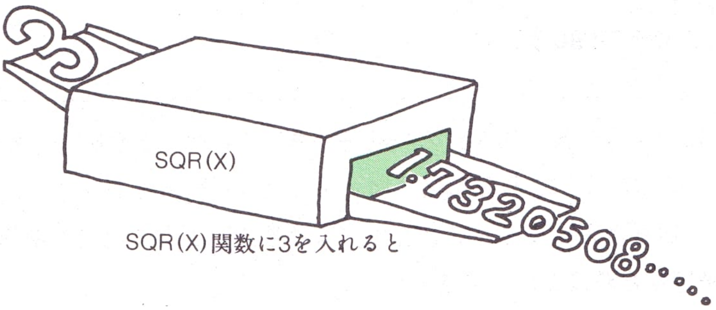
MSX_{2+}-BASICの関数のうち，数値を入れると数値が出てくる関数のことを数値型の関数といいます。
数値型の関数は全部で次の17種類あります。

ABS(X)　CDBL(X)　CINT(X)　CSNG(X)
FIX(X)　INT(X)　SQR(X)　SGN(X)
ATN(X)　COS(X)　SIN(X)　TAN(X)
LOG(X)　EXP(X)　RND(X)
ERR　ERL

平方根を求める SQR(X)

この中の，SQR(X)という関数を使って，関数の働きと使いかたをみていきましょう。SQR(X)という関数は，Xという数の平方根（ルート）を出してくれる働きをします。関数の話をするとき，出してくれる，というかわりに，「返す」ということばをよく使うので，そのいい方でいい直すと，**SQR(X)（スクエアルート）はXの平方根を返す関数**，ということになります。

たとえば，Xが3ならXの平方根は1.7320508……です。Xが5なら，Xの平方根は2.2360679……です。



SQR(X)関数に3を入れると

関数の使いかた

SQR(X)関数に3を入れると，1.7320508……が返ってきます。関数に値を入れるときは，Xを，入れる値にします。

```
SQR(3)
```

これで，SQR(X)関数に3が入ったことになります。もちろん，あらかじめ変数に3を代入しておいて，その変数を関数に入れることもできます。

```
A=3
```

を実行すれば，

```
SQR(A)
```

でSQR(X)関数に3が入ったことになります。

いずれの場合も，この時点で，SQR(X)関数は1.7320508……という値を持っていることになります。SQR(X)関数は，あなたが要求すればいつでも1.7320508……を返してくれます。

では，どうすれば関数は値を返してくれるのでしょうか。

関数自体は命令語ではありませんから，それだけでコンピューターに何かさせることはできません。関数から値を返してもらうためには，何かの命令語と組み合わせる必要があります。

たとえば，LET文やPRINT文が使えます。

```
R=SQR(3)
```

このLET文により，Rという数値型の変数にSQR(3)の値，すなわち1.7320508……が代入されます。ためしてみましょう。

```
R=SQR(3)
Ok
PRINT R
 1.7320508075688
Ok
```

小数点以下13ケタ，全部で14ケタの「3の平方根」の値が，変数Rに代入されました。

なお，MSX$_{2+}$-BASICでは，計算結果は14ケタの精度を持っています。これを，6ケタまでの精度に切り換えることもできます。有効数字14ケタの精度を倍精度，有効数字6ケタの精度を単精度と呼びます。

LET文を使わずに，PRINT文で直接関数の値を表示させることもできます。

```
PRINT SQR(5)
 2.2360679774998
Ok
```

また，IF—THEN文の条件式の中で関数を使うことも考えられます。

```
IF SQR(A)>=10 THEN END
```

Aの値によってSQR(A)の値も変わりますが，このIF—THEN文ではSQR(A)の値が10以上になったらプログラムを終了させることにしています。

このように，関数はそこに入れられた値を，その関数の持つ働きにしたがって一定のルールで加工した結果をつねに返します。返される値を利用するため，LET文，PRINT文やIF—THEN文などのBASICの命令語と組み合わせて使います。

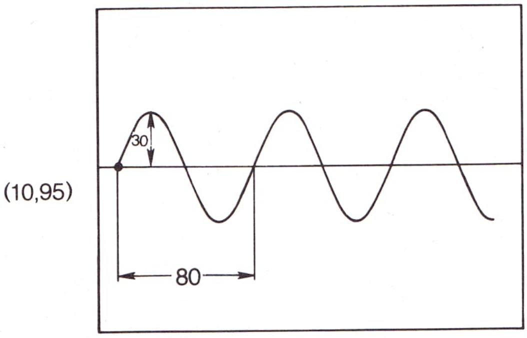
三角関数　SIN(X)

数値型の関数の中に，SIN(X)という関数があります。
これは，Xのサイン（正弦）を返す関数です。MSX_{2+}-BASICには，これ以外にCOS(X)，TAN（X），ATN（X）という三角関数があります。ここでは，SIN（X）を使って，サインカーブ（正弦曲線）を描くプログラムを作ってみます。

```
10 SCREEN 2:CLS
20 PI=3.14
30 LINE (0,95)-(252,95)
40 FOR X=10 TO 230
50 R=(X-10)*PI/40
60 S=SIN(R)
70 VY=S*30
80 Y=95-VY
90 PSET (X,Y)
100 NEXT X
110 GOTO 110
```

SIN(X)，COS(X)，TAN(X)を使うとき，Xの値はラジアンを単位とします。(1ラジアン＝π）そこで，このプログラムではXは10から230までの値をとるようにしていますが，それをいったんラジアンに変換しているのが行50です。Xの値の90を2ラジアン（2π），したがって50を1ラジアン（π）とし，Xが10のとき0ラジアンになるようにしています。(Xは，サインカーブを描くときの画面上でのX座標です。)
こうして求めたラジアン値のサインを，行60のSIN(X)関数で求め(S)，サインの値が1になったときのY軸方向の距離を30とし（行70），Y座標95を基準にサインカーブのY座標を求めます（行80）。

このプログラムを実行すると，画面上に次のようなサインカーブが描かれます。



絶対値を返す関数　ABS(X)

こんどはABS(X)という関数を例にとってみます。ABS(X)(アブソリュート)は，Xの絶対値を返す関数です。

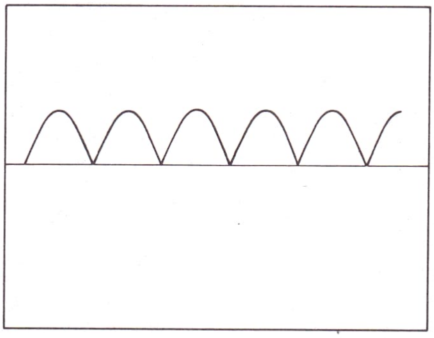
たとえば，Xが100のときABS(X)は100を返します。Xが−100のときも，ABS(X)は100を返します。

```
PRINT ABS(-3.5)
 3.5
Ok
PRINT ABS(-10)+ABS(8)
 18
Ok
```

このABS関数を，サインカーブのプログラムの中で使ってみましょう。

```
10 SCREEN 2:CLS
20 PI=3.14
30 LINE (0,95)-(252,95)
40 FOR X=10 TO 230
50 R=(X-10)*PI/40
60 S=SIN(R)
70 VY=ABS(S*30)
80 Y=95-VY
90 PSET (X,Y)
100 NEXT X
110 GOTO 110
```

行70を，VYに代入される値がSIN(R)＊30の絶対値となるように変更しました。その結果，描かれる図形は次のようになります。



ランダム関数　RND(X)

数値型の関数の中の，RND(X)（ランダム）という関数は，Xを0より大きい数にすると次々に0以上1未満の乱数を返す関数です。

```
PRINT RND(1)
```

を直接モードで繰り返し実行してみてください。

```
PRINT RND(1)
 .59521943994623
Ok
PRINT RND(1)
 .10658628050158
Ok
PRINT RND(1)
 .76597651772823
```

このように，RND(X)関数は1回実行されるごとに0以上1未満のでたらめな数を返します。でたらめな数のことを**乱数**といいます。RND(X)関数は乱数を発生する(返す)関数ということができます。

RND(X)関数を使って，ある範囲の値の乱数を発生させることができます。

整数に直す関数とRND(X)　INT(X)

RND関数を使って，1から200までの整数の乱数を発生させる方法を考えます。RND(1)自体は0以上1未満（0から0.99999999999999まで）の値を返します。そこで，RND(1)に200をかけます。

```
RND(1)*200
```

すると，この値は0以上200未満の数になります。ただし，値は小数点以下のケタを含む14ケタの数です。この値を整数に変換したいのですが，それには同じ数値型関数のINT(X)が使えます。

INT（インテジャー）(X)は，Xがどんな数であれ，それを整数にして返してくれます。

```
PRINT INT (12.73)
 12
Ok
PRINT INT (-7.99)
-8
Ok
```

このように，INT(X)関数はXより小さい最大の整数値を返します。

そこで，RND(1)＊200をINT(X)関数に入れ，

```
INT (RND(1)*200)
```

とします。すると，全体で0以上200未満（0から199まで）の値を乱数として返すことになります。したがって，0以上200以下の乱数を返すようにするには，

```
INT ((RND(1)*(200+1))  すなわち
INT (RND(1)*201)
```

とすれば良いことになります。

```
10 FOR L=1 TO 10
20 X=INT(RND(1)*201)
30 PRINT X;
40 NEXT L
```

このプログラムを実行すると，0から200までの乱数を10回発生して表示します。

```
RUN
 119  21  153  116  147  37  74  190
 128  94
```

一般に0以上N以下の乱数を発生させるときは

X＝INT(RND(1)＊(N＋1))

という形を使います。これにより，Xに0以上N以下の乱数が代入されます。

これを応用して，M以上N以下の乱数を発生させるには，

X＝INT(RND(1)＊(N－M＋1))＋M

という形が使えます。2以上15以下の乱数を発生させるなら，M＝2，N＝15ですから，N－M＋1＝14，したがって

X＝INT(RND(1)＊14)＋2

となります。

ランダムボックス

RND(X)関数を使って，大きさ，色，表示位置がまちまちな正方形を50個描くプログラムを作りましょう。

```
10 SCREEN 5
20 COLOR ,1,1:CLS
30 FOR B=1 TO 50
40 SX=INT(RND(1)*220)+5
50 SY=INT(RND(1)*180)+5
60 ST=INT(RND(1)*40)+20
70 C=INT(RND(1)*14)+2
80 LINE (SX,SY)-STEP(ST,ST),C,BF
90 NEXT B
100 GOTO 100
```

正方形の左上の頂点の座標をSX, SY，1辺の長さをST，カラーコードをCという変数であらわすことにし，行40～70でRND(X)関数によりこれらの変数に乱数を代入しています。乱数の値の範囲はそれぞれ次のとおりです。

SX ……5以上224以下

SY ……5以上184以下

ST ……20以上59以下

C ………2以上15以下

そして，行80により大きさ，色，表示位置の異なる正方形が次々と描かれます。

プログラムの実行ごとに異なる乱数を発生させるには

さきほどのランダムボックスのプログラムを何回か実行してみるとわかるとおり，いつも同じ位置に四角形が描かれます。実は，コンピューターは，乱数のリストを内部に持っているのですが，RND(1)関数を使ったプログラムを実行すると，実行するたびにこの乱数のリストの最初の値から順にRND(1)関数によって返されるのです。そこで，プログラムを実行するたびに，乱数のリストの異なる部分の値が返されるように工夫すると，実行ごとに違う結果が得られるようになります。

ひとつの方法は，内蔵タイマーを利用して次のような行をプログラムの最初に置きます。

```
22 FOR N=0 TO TIME-INT(TIME/100)*100
24 X=RND(1)
26 NEXT N
```

行22で使われているTIMEはMSX-BASICの特別な変数で常に内蔵タイマーの値が自動的に代入され続けます。内蔵タイマーは0から65535まで，約1／60秒ごとに1ずつ繰り上がっていきます。そして65535になると次は再び0になります。

上に示したような行を持つプログラムを実行したときのTIMEの値は0から65535までのいずれかです。たとえば，実行したときの内蔵タイマーの値が42280だったとします。行22には2つのTIME変数が使われていますが，内蔵タイマーの値は常に速いスピードで変化していますから，最初のTIME変数と次のTIME変数に代入される値には少し差ができます。最初のTIME変数に42280が代入されたとすると，次のTIME変数には42281あたりの値が代入されます。そこで，行22全体としては

FOR　N＝0　TO　42280－INT(42281／100)＊100

すなわち

FOR　N＝0　TO　80

ということになります。

あるいは，最初のTIME変数に10900が，次のTIME変数に10901が代入されたとすると

FOR　N＝0　TO 10900－INT（10901／100）＊100

すなわち

FOR　N＝0　TO　0

になります。

このように，プログラムを実行したときのFOR—NEXT文におけるNの最終値はそのときの内蔵タイマーの値によって変化します。Nの最終値が80のときは，FOR—NEXTループにより行24のX＝RND(1)が81回実行されます。したがって，この次にRND(1)関数が使われると，このRND(1)関数は乱数リストの82番目の値を返すことになります。

では，先のランダムボックスのプログラムに，この3行を追加してみましょう。

```
10 SCREEN 5
20 COLOR ,1,1:CLS
22 FOR N=0 TO TIME-INT(TIME/100)*100
24 X=RND(1)
26 NEXT N
30 FOR B=1 TO 50
40 SX=INT(RND(1)*220)+5
50 SY=INT(RND(1)*180)+5
60 ST=INT(RND(1)*40)+20
70 C=INT(RND(1)*14)+2
80 LINE (SX,SY)-STEP(ST,ST),C,BF
90 NEXT B
100 GOTO 100
```

プログラムの実行ごとに異なる乱数を得る方法として，もうひとつ挙げておきましょう。これは，キーボードからのデータ入力の関数であるINKEY$を使うものです。

やはり，次の3行をプログラムに追加して使います。

```
22 A$=INKEY$
24 X=RND(1)
26 IF A$="" THEN 22
```

INKEY$関数については，187ページに詳しい説明があります。この3行により，キーボード上のキーが何かひとつ押されるまで，いつまでも行24のX＝RND(1)が繰り返し実行されることになります。

ランダムボックスのプログラムに，この3行を追加して試してみてください。

文字型の関数

- 文字型関数とは
- 文字列の加工
- 日本語文字列の処理

文字型関数とは

ある文字列を入れると，その文字列を加工して，その結果の文字列を返す関数を文字型関数といいます。

MSX$_{2+}$-BASICには，次の7種類の文字型関数があります。

LEFT$(X$, N)，MID$(X$, M, N)，RIGHT$(X$, N)，SPACE$(N)，
STRING$(N, J)またはSTRING$(N, X$)，TAB(N)，SPC(N)

この中のいくつかは，数値を入れると文字列を返す関数ですが，文字型関数のグループに入れておきます。

必要な数だけスペースを　SPACE$(N)，SPC(N)

文字型関数のうち，SPACE$(N)（スペースダラー）およびSPC(N)（スペース）は，N字分のスペースを返す関数です。どちらも働きはまったく同じですが，SPC(N)の方はPRINT文だけでしか使えない，という制約があります。

直接モードで，これらの関数の働きをみてみましょう。

```
PRINT "A";SPACE$(10);"B"
A          B
Ok                         10個のスペース
PRINT "C";SPC(15);"D"
C               D
Ok                         15個のスペース
S$=SPACE$(5)
Ok
PRINT "X";S$;"Y";S$;"Z"
X     Y     Z
Ok                         5個のスペース
```

文字型関数は，つねに文字を返します。したがって，その値をLET文で代入するとき，変数も必ず文字型でなければなりません。上の例では，

```
S$=SPACE$(5)
```

により，文字型変数S$に5個のスペースを代入しています。

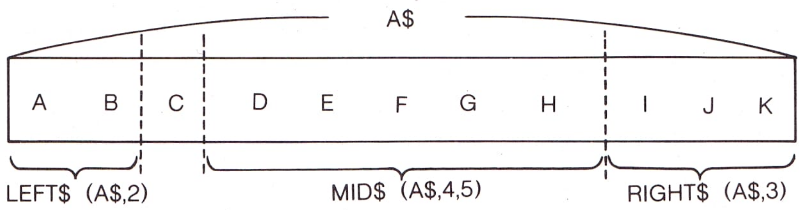
文字列の加工
LEFT$(X$, N), MID$ (X$, M, N), RIGHT$ (X$, N)

文字型関数のうち，LEFT$(X$, N), MID$(X$, M, N), RIGHT$ (X$, N) は，与えられた文字列の一部分を返す関数です。

LEFT$(レフトダラー)(X$, N)は，文字列X$の左からN文字だけを分離して返します。
RIGHT$(ライトダラー)(X$, N) は，X$の右からN文字だけを分離して返します。
MID$(ミッドダラー) (X$, M, N) は，X$のM文字目以降のN文字だけを分離して返します。
実例で確認しましょう。

```
A$="ABCDEFGHIJK"
Ok
PRINT LEFT$(A$,2)
AB
Ok
PRINT RIGHT$(A$,3)
IJK
Ok
PRINT MID$(A$,4,5)
DEFGH
Ok
```

このように，LEFT$(X$, N)，RIGHT$(X$, N)，MID$(X$, M, N)の各関数を使うと，指定した文字列の中から任意の部分だけを取り出すことができます。



次のプログラムは，LEFT$(X$, N)関数を使って，INPUT文で入力された文字列(人の名前)を，全部表示したあと，Jで始まる名前，つまり左の1文字がJの文字列だけを別に表示させるものです。

```
10 DIM N$(10):Y=1
20 CLS
30 FOR L=1 TO 10
40 INPUT "Name";N$(L)
50 NEXT L
60 CLS
70 FOR L=1 TO 10
80 LOCATE 2,L:PRINT N$(L)
90 NEXT L
100 FOR L=1 TO 10
110 A$=LEFT$(N$(L),1)
120 IF A$<>"J" THEN 150
130 LOCATE 15,Y:PRINT N$(L)
140 Y=Y+1
150 NEXT L
160 LOCATE 0,20
```

行40のINPUT文で，配列変数N$(1)～N$(10)に人の名前を代入していきます。たとえば，次の名前を順番にN$(1)から代入したとします。

PETER，PAUL，JACK，MARY，SUSIE，JOHN，
JANE，TOM，DICK，CATHARINE

すると，行70～90により，代入されたすべての名前が表示されたあと，行100～150によりJで始まる名前，つまりJACK，JOHN，JANEだけが画面の右側に表示されます。

```
PETER        JACK
PAUL         JOHN
JACK         JANE
MARY
SUSIE
JOHN
JANE
TOM
DICK
CATHARINE
```

Jで始まる名前だけを選び出すために，行110でLEFT$ (X$, N) 関数が使われています。ここで，A$に配列変数N$に代入されている文字列の一番左の文字だけを取り出して代入し，行120のIF—THEN文によりそれが“J”かどうかによってN$の内容を表示するかどうか決めています。

日本語文字列の処理　CALL KMID（A$, X$, M, N）など

CALL KANJI文で日本語入力モードにしてあるときは，MID$（X$, M, N）などの文字型関数は使えなくなります。そのかわり，MSX_{2+}-BASICでは，日本語入力モードで同様の処理ができるようにするための命令語が準備されています。
ここでは，そのひとつであるCALL KMID文を例にとって説明します。

CALL KMID（A$, X$, M, N）文は，X$のM文字目以降のN文字だけを分離して，文字型変数A$に代入します。つまり，日本語入力モード以外でMID$関数を使い，A$＝MID$（X$, M, N）を実行したのと同じ働きをしてくれます。
実例をみてみましょう。

```
CALL KANJI
Ok
X$="日本語文字列"
Ok
CALL KMID(A$,X$,4,2)
Ok
CALL KMID(B$,X$,1,3)
Ok
PRINT B$;"の";A$
日本語の文字
```

日本語入力モードでは，CALL KMID文をはじめとして，次のような命令語によって，日本語の文字列を処理することができます。

命令語	働き
CALL KMID（A$, X$, M, N）	X$のM文字目以降N文字を分離してA$に代入する。
CALL KINSTR (A, N, X$, Y$)	X$のN文字目以降にY$があるかを探して，その位置をAに代入する。(Y$が見つからなければ0が代入される。)

命令語	働き
CALL KLEN（A, X$, N）	Nが0（または省略）のとき： 　X$の文字数をAに代入する。 Nが1のとき： 　X$の中の半角文字の数をAに代入する。 Nが2のとき： 　X$の中の全角文字の数をAに代入する。
CALL JIS（A$, X$） CALL SJIS（A$, X$）	X$の先頭の文字の文字コード(16進4けた)を，文字列としてA$に代入する。 CALL JIS文の場合，文字が日本語のときはそのJISコードが，CALL SJIS文の場合そのシフトJISコードが代入される。
CALL KNJ（A$, X$）	X$の16進4けたの文字コードに対応する文字をA$に代入する。
CALL AKCNV（A$, X$）	X$中のすべての文字を全角文字に変換してA$に代入する。
CALL KACNV（A$, X$）	X$の中で半角文字に変換できる文字を半角文字に変換して，全体をA$に代入する。
CALL KEXT（A$, X$,N）	Nが0のとき： 　X$の中の半角文字だけを分離してA$に代入する。 Nが1のとき： 　X$の中の全角文字だけを分離してA$に代入する。
CALL KTYPE (A, X$, N)	X$のN番目の文字が半角なら0を，全角なら1をAに代入する。

数値型・文字型変換の関数

- 数値型・文字型変換の関数のいろいろ
- 文字コードについて

数値型・文字型変換関数のいろいろ

数値型・文字型変換の関数にはいろいろな種類があり，その働きもさまざまですが，共通しているのは数値型の値を入れると文字型の値を返す関数であるか，文字型の値を入れると数値型の値を返す関数であるかのいずれか，ということです。

MSX_{2+}-BASICには，次の9種類の数値型・文字型変換の関数があります。

ASC(X$)，CHR$(X)
VAL(X$)，STR$(X)
LEN(X$)，INSTR(N, X$, Y$)
BIN$(X)，OCT$(X)，HEX$(X)

数字の型を変える　VAL(X$)，STR$(X)

BASICでは，数字は数値型，文字型のいずれでも扱うことができます。
たとえば，123という数字を

```
A=123
```

のように数値型変数に代入すれば，123をヒャクニジュウサンという数値として扱ったことになります。また，

```
A$="123"
```

のように文字型変数に代入すれば，123はイチニサンという文字として扱ったことになります。
文字として扱った数字を数値型に変換するのがVAL（バリュー）(X$)，数値として扱った数字を文字型に変換するのがSTR$（エスティーアールダラー）(X)という関数です。
次のプログラムで確かめてみましょう。

```
10 A$="123":B$="456"
20 X=VAL(A$):Y=VAL(B$)
30 PRINT "A$+B$=";A$+B$
40 PRINT "VAL(A$)+VAL(B$)=";X+Y
```

123および456という数字を，まず文字型の値として変数A$, B$に代入し，それを行20で数値型に変換しています。文字型，数値型，それぞれの場合の加算を行って表示させているのが行30，40です。実行結果は次のようになります。

```
RUN
A$+B$=123456
VAL(A$)+VAL(B$)= 579
Ok
```

VAL(A$)と逆の働きをするのがSTR$(X)です。

次のプログラムで確認してください。

```
10 A=123:B=456
20 X$=STR$(A):Y$=STR$(B)
30 PRINT "A+B=";A+B
40 PRINT "STR$(A)+STR$(B)=";X$+Y$
RUN
A+B= 579
STR$(A)+STR$(B)= 123 456
Ok                 X$   Y$
```

STR$(X)関数で数値型の数を文字型に変換すると，数値型の数の前にあるスペース（+，−の記号が入る場所）もいっしょに文字型の値の中に組み込まれます。

文字コードと関数　ASC(X$), CHR$(X)

色にカラーコードがあるように，BASICで使える文字には文字コード（キャラクターコードとも呼びます）というものがあります。
たとえば，大文字のAの文字コードは65（10進数）です。MSX2+-BASICには，文字コードを扱う関数がふたつあります。ASC（アスキー）(X$)は，文字を入れるとその文字コードを返します。CHR$（キャラクターダラー）(X)は，文字コードを入れるとその文字を返します。
直接モードで確かめましょう。

```
PRINT ASC("A")      文字Aの文字コードを
 65                 表示
Ok
PRINT CHR$(66)      文字コード66の文字
B                   (B)を表示
Ok
```

文字列の長さを返す　LEN(X$)

LEN(X$)関数は，文字列X$の文字数を数値型データとして返す関数です。

```
A$="ABCDE"
Ok
PRINT LEN(A$)
 5
Ok
```

LEN(X$)により，A$に代入された文字列 "ABCDE"の文字数である5が返されました。

次のプログラムは，LEN(X$)，MID$(X$, M, N)，CHR$(X)の関数を使って，INPUT文で変数に代入した各文字を，それぞれ文字コードがひとつ大きい文字に変換して表示するようにしたものです。

```
10 CLS
20 INPUT "Any letters";A$
30 N=LEN(A$)
40 FOR L=1 TO N
50 B$=MID$(A$,L,1)
60 X=ASC(B$)
70 C$=CHR$(X+1)
80 PRINT C$;
90 NEXT L
100 END
```

何回か実行してみてください。

```
RUN
Any letters? RNMX
SONY
Ok
RUN
Any letters? LRW
MSX
Ok
```

データ入力の関数

- データ入力関数のいろいろ
- キーボードからデータ入力
- カーソルキーの状態を入力

データ入力関数の働き

今までみてきた関数は，関数に入れた値をその関数の持つ働きによって加工して返すものでした。
ここで扱う関数はちょっとタイプが異なります。関数に入る値は，直接加工するために入れるのではなく，ある意味を持つ符号のようなものです。そして，データ入力関数と呼ばれる関数はその符号の示す意味にしたがって主にコンピューターに接続された機器の状態を入力して，その状態をデータとして返してくれます。(中には，まったく値を入れる必要のない関数もあります。)
これらの関数を使うことで，接続機器の状態によって何かを行うプログラムの作成が可能になります。
たとえば，カーソルキーが押されるとその方向にスプライトパターンが働くプログラムなどが考えられます。

データ入力関数にはどんなものがあるか

MSX$_{2+}$-BASICには，22のデータ入力関数があります。どんなものがあるかだけを並べ挙げておきましょう。

●スクリーンから入力

CSRLIN，POS(X)，POINT(X,Y)

●プリンターから入力

LPOS(X)

●メモリーから入力

FRE(0)，FRE("")，PEEK(N)，VARPTR(変数)，VPEEK(N)

●キーボードから入力

INKEY$，INPUT$(X)

●I/Oポートから入力

INP(N)

●ジョイスティック，スペースバー，マウス，トラックボール，パドル，タッチパッド，ライトペンから入力

STICK(N)，STRIG(N)，PDL(N)，PAD(N)

●データファイルから入力

EOF(ファイル番号)，INPUT$(N,#ファイル番号)

●ディスクから入力

DSKF，LOC，VARPTR(#ファイル番号)

●機械語サブルーチンから入力

USR(X)

キーボードからデータ入力　INKEY$

INKEY$(インキーダラー)関数は，キーボード上のキーがひとつ押されると，押されたキーの文字を文字型のデータとして返します。

簡単なプログラムで，INKEY$の働きを調べましょう。

```
10 K$=INKEY$
20 PRINT K$;
30 GOTO 10
```

このプログラムを実行すると，何度も行10の

　　　　　K$=INKEY$

が実行されます。行10が実行されたときに何もキーが押されていないと，INKEY$は何も返しません。あるいは，文字列として何もない状態のことをヌルストリングと呼ぶので，ヌルストリングを返す，ということもできます。つまり，K$には何も代入されません。(あるいは，ヌルストリングが代入されます。)
したがって，行20のPRINT文では，何もないヌルストリングを表示することになるので，見かけ上は何も起こりません。
プログラム実行中に，何かキーを押したとします。たとえば，[A]キーを押したとすると，ここでINKEY$はAという文字を返します。(小文字になるように押した場合はaを返します。) それが，K$に代入され，行20で表示されるので，画面上にAが表示されます。

続けてキーを押していくと，押したキーの文字が次々に表示されていきます。一見，コマンド待ちの状態と似ていますが，カーソルが消えているのでプログラム実行中であることが分かります。また，[RETURN]キーを押しても改行されず，同じ行の先頭から次の文字が表示されるのもこのプログラムの働きによります。
プログラムを止めるには，[CTRL]+[STOP]キーを使います。

では，INKEY$を使って，もうひとつプログラムを作ってみましょう。

```
10 CLS
20 LOCATE 5,4
30 PRINT "┌────────────────┐"
40 LOCATE 5,14
50 PRINT "└────────────────┘"
60 FOR Y=5 TO 13 STEP 2
70 FOR X=6 TO 20
80 K$=INKEY$
90 IF K$="" THEN 80
100 LOCATE X,Y:PRINT K$
110 NEXT X
120 NEXT Y
130 LOCATE 0,22:END
```

このプログラムでは，INKEY$の重要な使いかたが示されています。
行80のINKEY$と行90のIF-THEN文の組み合わせに注目してください。

```
80 K$=INKEY$
90 IF K$="" THEN 80
```

行80が実行されたときに何もキーが押されていないと，K$にはヌルストリングが代入されます。行90のIF-THEN文は，K$の内容がヌルストリングなら行80に戻しています。つまり，何もキーを押さない限り，プログラムはここでループを繰り返すことになります。

```
K$=""
```

の"" は，"（ダブルクォート）の間になにもはさまず（スペースも），これでヌルストリングをあらわしています。そして，何かキーが押されると行100に進み，押されたキーの文字が表示されることになります。

このINKEY$関数の使いかたは，何かのキーを押すことによって次のステップに進むようなプログラムを作るときに多く利用されます。

さらに一歩進んで，何か特定のキーを押すと次のステップに進むようなプログラム(たとえば，スペースバーを押すと何かが起こるプログラム)を作るときは，INKEY$関数を次のように使うと良いでしょう。

```
10 CLS
20 INPUT "Any letters";A$
30 K$=INKEY$
40 IF K$="" OR K$<>" " THEN 30
50 PRINT A$
```

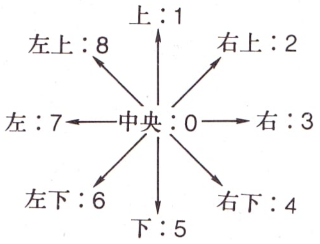
行40のIF-THEN文により，①K$がヌルストリングのとき，または，②K$がスペースではないとき，は行30に戻ります。そして，スペースバーが押されたときだけ行50に進むことになります。

カーソルキーの状態を入力　STICK(N)

STICK(N)関数は，カーソルキー，ジョイスティック，マウス，トラックボールおよびタッチパッドの方向を示す数値を返す関数です。
カーソルキーの状態を入力するか，ジョイスティックなどの状態を入力するかは，Nの値によって決まります。

N	何の状態か
0	カーソルキー
1	CONTROLLER A端子に接続した装置
2	CONTROLLER B端子に接続した装置

また，STICK(N)関数が返す値は0から8までで，カーソルキーまたはジョイスティックなどの示す方向をあらわします。



たとえば，STICK(0)は，どのカーソルキーも押されていなければ0を返します。(上方向)が押されると1を，(下方向)が押されると5を返します。また，とを同時に押すと2を返します。

このSTICK(N)関数を使って，カーソルキーにより画面表示をコントロールするプログラムを作ってみます。

```
10 CLS
20 X=14:Y=10
30 LOCATE X,Y:PRINT "o"
40 C=STICK(0)
50 IF C=0 THEN 40
60 IF C=1 THEN VX=0:VY=-1:GOSUB 110
70 IF C=3 THEN VX=1:VY=0:GOSUB 110
80 IF C=5 THEN VX=0:VY=1:GOSUB 110
90 IF C=7 THEN VX=-1:VY=0:GOSUB 110
100 GOTO 40
110 X=X+VX:Y=Y+VY
120 IF X>29 THEN X=29
130 IF X<0 THEN X=0
140 IF Y>21 THEN Y=21
150 IF Y<0 THEN Y=0
160 LOCATE X,Y:PRINT "o"
170 RETURN
```

このプログラムは，o(小文字のオー)を，カーソルキーの方向にしたがって，その方向に次々と表示していくものです。最初，oは14，10の位置に表示されます。(行20，30)。カーソルキーの状態を調べているのが行40のSTICK(0)関数で，STICK(0)が返す値によって，oを次に表示する位置を決めるための変数VX，VYの値を決定しているのが行50～90です。たとえば，▷ を押すとSTICK(0)は3を返しますから，行70によりVXは1，VYは0となって行110からのサブルーチンへ行きます。
サブルーチンで，次にoを表示する位置をX，Yに代入してoを表示します。
サブルーチン内では，X，Yの値が一定の範囲を越えると，上限，下限の値にとどまるよう行120～150で調整しています。

第9章
割り込み

割り込みをかけるには

- ●割り込みとは
- ●MSX-BASICの割り込み
- ●割り込みをかけるには

割り込みとは

割り込みとは，プログラムの実行中に，特定の条件が発生したときに，それまで実行していたプログラムの流れを中断して別の処理を行うことをいいます。割り込みが起こったときに実行される処理プログラムを割り込み処理プログラムまたは割り込み処理ルーチンと呼びます。

割り込みと良く似た考え方に，サブルーチンがありますが，サブルーチンは，MSX-BASICではGOSUB文が実行されたときだけ実行されるものです。つまり，いつサブルーチンが実行されるかは，あらかじめプログラム内部で決められています。
それに対して，割り込み処理ルーチンの方は，プログラムの外の条件（例えば[F1]キーが押されたときなど）により実行されるものです。
割り込み処理ルーチンの実行が終わると，サブルーチンの場合と同じように，通常はもとのプログラムの実行を再開するようにします。

MSX-BASICで使える割り込み

MSX-BASICでは，割り込みをかける外部条件として，次の5種類が使えます。

- ●ファンクションキー（[F1]-[F5]）のいずれかが押されたとき。
- ●スペースバーまたはジョイスティック，マウス，トラックボール，タッチパッドのボタンが押されたとき。
- ●[CTRL]＋[STOP]キーが押されたとき。
- ●スプライトどうしが衝突したとき。
- ●一定時間が経過したとき（内蔵タイマーによる）。

割り込みをかけるには

割り込みの宣言

MSX-BASICのプログラムを実行中に割り込みがかけられるようにするには，プログラムのメインルーチンであらかじめ5種類の割り込み宣言文のいずれかを実行します。割り込み宣言文の次に，割り込みを有効にする文を実行します。割り込みを有効にする文も，使う割り込みによって次の5種類があります。

割り込み	割り込み宣言文	割り込みを有効にする文
ファンクションキーによる割り込み	ON KEY GOSUB 行番号［，行番号］…	KEY（N）ON Nは1～10　1のとき F1 キー
スペースバー，ジョイスティック，マウス，トラックボール，タッチパッドのボタンによる割り込み	ON STRIG GOSUB 行番号［，行番号］…	STRING(N) ON Nは0～4　0のときスペースバー
CTRL ＋ STOP による割り込み	ON STOP GOSUB 行番号	STOP ON
スプライト衝突による割り込み	ON SPRITE GOSUB 行番号	SPRITE ON
内蔵タイマーによる割り込み	ON INTERVAL＝インターバル時間 GOSUB 行番号	INTERVAL ON

割り込み宣言文は，何によって割り込みをかけるのかを宣言し，割り込みがかかったときに実行される割り込み処理ルーチンの先頭の行番号を指定します。

割り込みを使ったプログラム

- ファンクションキーによる割り込み例
- 割り込みを無効にする
- 割り込みを保留にする
- スプライト衝突による割り込み例

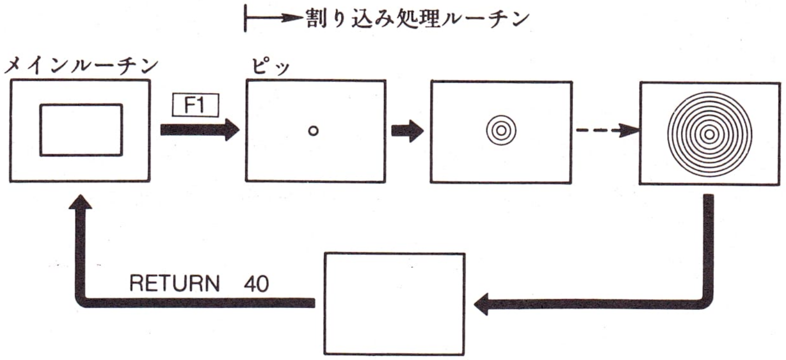
ファンクションキーによる割り込みプログラム

次のプログラムは，[F1]キーによる割り込みを試してみるためのものです。

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'subroutine
110 BEEP:CLS
120 FOR L=10 TO 90 STEP 10
130 CIRCLE (120,100),L
140 NEXT L
150 CLS
160 RETURN 40
```

行10〜50：メインルーチン
行100〜160：割り込み処理ルーチン

このプログラムでは，行10，20により，[F1]キーが押されたら行100からの割り込み処理ルーチンに移行するように設定しています。
プログラムを実行すると，メインプログラムの行40，50により常に長方形が表示され続けますが，[F1]キーを押すと割り込みがかかり，行100へ移行します。それにより，ピッという音とともに長方形は消え(BEEP：CLS)，続いて9つの円が次々と描かれていきます。最後の円が描かれると，画面はクリアされ，再び行40へ戻ります。



割り込みを無効にする　KEY(N) OFF

上のプログラム例で，次の行を追加してみます。

```
105 KEY(1) OFF
```

この行を追加したプログラムを実行すると，[F1]キーを1回目に押したときは割り込みがかかりますが，それ以降は[F1]キーを押しても割り込みはかからなくなります。これは，最初に割り込み処理ルーチンが実行されたときに行105の

```
KEY(1) OFF
```

が実行されたためで，これにより[F1]キーによる割り込みが無効になったのです。

KEY(N) OFFは，ファンクションキーによる割り込みを無効にする文です。Nでファンクションキー番号を指定します。

MSX-BASICの5種類の割り込みと，割り込みを無効にする文は次のとおりです。

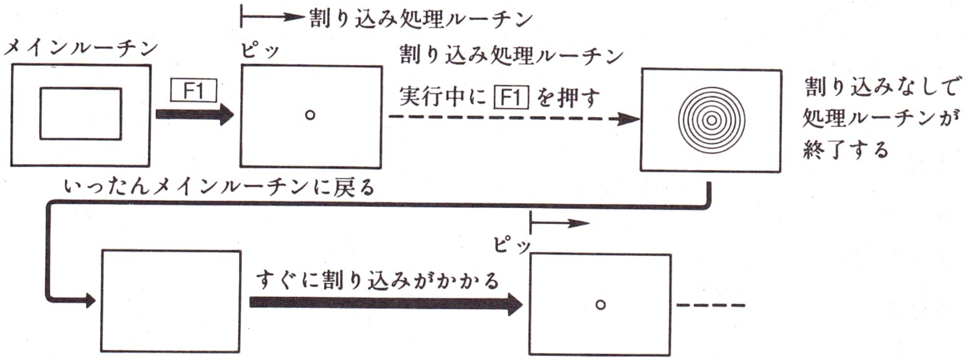
割り込み	割り込みを無効にする文
ファンクションキーによる割り込み	KEY(N) OFF
スペースバー，ジョイスティック，マウス，トラックボール，タッチパッドのボタンによる割り込み	STRIG(N) OFF
[CTRL]＋[STOP]による割り込み	STOP OFF
スプライト衝突による割り込み	SPRITE OFF
内蔵タイマーによる割り込み	INTERVAL OFF

割り込みの保留

割り込みにより，実行が割り込み処理ルーチンに移ると，他の割り込みは保留の状態になります。この状態では，もう1度割り込みをかけてもすぐに割り込みはかからず，いったんRETURN文によりメインルーチンに戻りますが，戻った時点で自動的に-ONの状態になり，メインルーチンは実行されずにすぐに割り込み処理ルーチンに移行します。

つまり，割り込み保留の状態では，割り込みをかけても割り込み処理ルーチンの最初に戻ることはなく，ただ割り込みがかかったことだけが覚えられ，処理ルーチンからいったん抜け出してから割り込みがかかることになります。

194ページのプログラムの場合，F1キーを押すと割り込みにより9つの円が描かれますが，最後の円が描かれる前にF1キーを押してもこの時点では割り込みはかかりません。そして，最後の円が描かれるとメインルーチンに戻りますが，ここで2回目に押したF1キーによる割り込みがかかり，長方形は描かれず，再び円が描かれます。



割り込み処理ルーチンで さらに割り込みを有効にする　KEY(N) ON

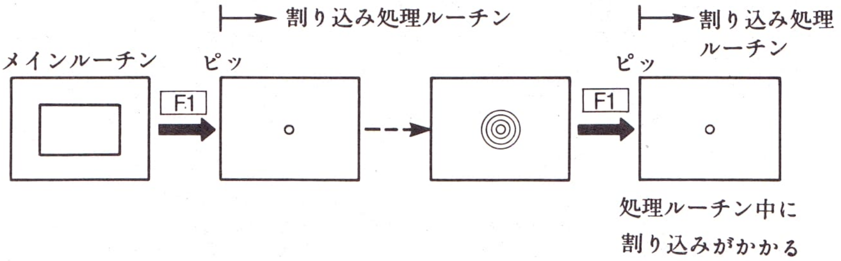
割り込み処理ルーチン実行中，さらに同じ割り込みを有効にするには，割り込み処理ルーチン中にKEY(1)ONなどのコマンドを置きます。これにより，割り込み処理ルーチンに入ってからも同じ割り込みをかけて，もう1度割り込み処理ルーチンを最初から実行させることができるようになります。

次のプログラムをみてください。

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'subroutine
105 KEY(1) ON
110 BEEP:CLS
120 FOR L=10 TO 90 STEP 10
130 CIRCLE (120,100),L
140 NEXT L
150 CLS
160 RETURN 40
```

先のプログラムと同じですが，行105にKEY(1)ONという文を置いています。

これにより，割り込みがかかって次々と円を描いている途中でさらに[F1]キーを押すと，すぐに割り込みがかかり，再び行100からの割り込み処理ルーチンが最初から実行されます。

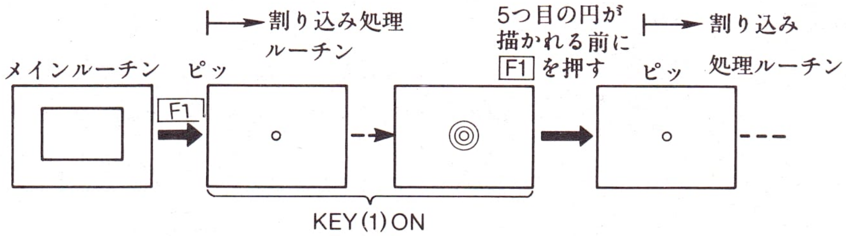


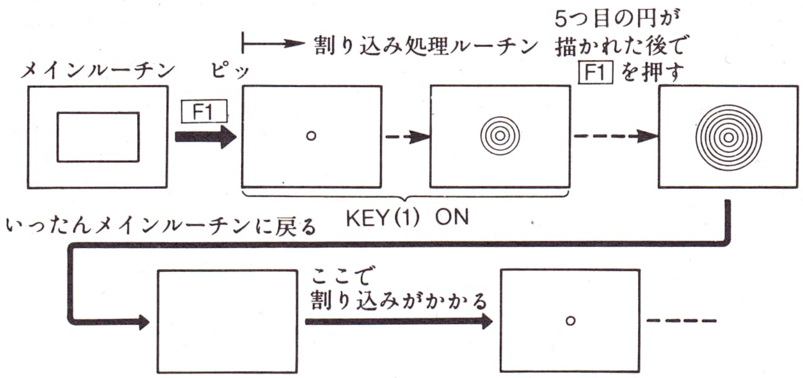
プログラムで割り込みを保留にする　KEY(N) STOP

割り込み処理ルーチンで，KEY(N)ON文により割り込みを有効にしたあとで再び保留の状態にするには，KEY(N)STOP文を使います。

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'subroutine
105 KEY(1) ON
110 BEEP:CLS
120 FOR L=10 TO 90 STEP 10
130 CIRCLE (120,100),L
135 IF L=50 THEN KEY(1) STOP
140 NEXT L
150 CLS
160 RETURN 40
```

このプログラムも先のものと同様ですが，行135で，Lの値が50になったらKEY(1)STOPが実行されるようにしています。この結果，割り込み処理ルーチン実行中でも，5つ目の円が描かれる前ならば F1 キーによりすぐに割り込みがかかりますが，5つ目の円が描かれると割り込みは保留となり，それ以後 F1 を押してもすぐに割り込みはかからなくなります。





KEY(N)STOPは，ファンクションキーによる割り込みを保留にします。Nでファンクションキー番号を指定します。MSX-BASICの5種類の割り込みと，割り込みを保留にする文は次のとおりです。

割り込み	割り込みを保留にする文
ファンクションキーによる割り込み	KEY(N) STOP
スペースバー，ジョイスティック，マウス，トラックボール，タッチパッドのボタンによる割り込み	STRIG(N) STOP
[CTRL]+[STOP]による割り込み	STOP STOP
スプライトの衝突による割り込み	SPRITE STOP
内蔵タイマーによる割り込み	INTERVAL STOP

スプライト衝突による割り込み

ON SPRITE GOSUB文およびSPRITE ONにより，2つ以上のスプライトパターンが1ドットでも重なったら割り込みを発生させることができます。

次のプログラムでは，左右からUFOが飛んできて，UFOが重なっている間ピッという音が鳴り続けます。

```
10 SCREEN 2
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 ON SPRITE GOSUB 100
40 SPRITE ON
50 FOR X=0 TO 255
60 PUT SPRITE 0,(X,100),15,0
70 PUT SPRITE 1,(255-X,100),10,0
80 NEXT X
90 END
100 SPRITE OFF
110 BEEP
120 SPRITE ON
130 RETURN
```

サンプルプログラム

次のプログラムは，F1キーによる割り込みとスプライト衝突による割り込みを使った，アーチェリーゲームのプログラムです。F1キーを押すと矢が放たれます。

矢が的に当たると，当たった位置によって得点が決まります。矢は5回射ることができます。得点が1000点以上になるともう1回繰り返してプレイできます。

```
10 SCREEN 5,1
20 OPEN "GRP:" FOR OUTPUT AS #1
30 ON KEY GOSUB 250 : ON SPRITE GOSUB 27
0
40 ' *** sprite definition ***
50 RESTORE 430:SN=0:GOSUB 310      ┐
60 RESTORE 520:SN=1:GOSUB 310      ┘ スプライト定義
70 ' *** main routine ***          ┐
80 X=230:S=0:W=50                  │
90 FOR L=1 TO 5                    │
100  SPRITE  ON:KEY (1) ON         │
110  U=W:M=0:B=0:V=106:GOSUB 210   │ 矢を表示，
120  FOR Y=0 TO 211                │ 的を上から
130   PUT SPRITE 0,(X,Y),,0        │ 下へ動かす
140   U=U+B:IF M=1 THEN V=V+1      │
150   PUT SPRITE 1,(U,V),,1        │
160   IF U>250 THEN U=W:B=0        │
```

```
170  NEXT Y
180 NEXT L
190 IF S>=1000 THEN 80 ELSE END
200 ' *** score display ***
210 PRESET (30,10)
220 PRINT #1,USING "## : ####";L,S
230 RETURN
240 ' *** shoot ***
250 KEY (1) OFF:B=3:RETURN
260 ' *** hit ***
270 SPRITE OFF:B=0:M=1
280 S=S+(8-ABS(Y-V+1))^2*10
290 GOSUB 210 : RETURN
300 ' *** sprite definition subroutine *
**
310 SP$="":SC$=""
320 FOR SL=0 TO 7
330 READ SQ$,SC:SP=0
340  FOR SJ=1 TO 8
350   SP=SP*2-(MID$(SQ$,SJ,1)="O")
360  NEXT SJ
370  SP$=SP$+CHR$(SP):SC$=SC$+CHR$(SC)
380 NEXT SL
390 SPRITE$(SN)=SP$
400 COLOR SPRITE$(SN)=SC$
410 RETURN
420 ' *** sprite data ***
430 DATA ......OO,1
440 DATA ......OO,8
450 DATA ......OO,15
460 DATA ......OO,8
470 DATA ......OO,8
480 DATA ......OO,15
490 DATA ......OO,8
500 DATA ......OO,1
510 ' *** sprite data ***
520 DATA ........,0
530 DATA ........,0
540 DATA ........,0
550 DATA OOOOOOOO,15
560 DATA ........,0
570 DATA ........,0
580 DATA ........,0
590 DATA ........,0
```

- 210–230: 得点表示サブルーチン
- 250: F1 が押されたら，Bに3を代入（矢が動く）
- 270–290: スプライトが衝突したら，得点計算
- 310–410: スプライト定義サブルーチン
- 420–500: スプライトパターンデータ（的）
- 510–590: スプライトパターンデータ（矢）

行20のOPEN文は、グラフィックモードの画面にキャラクターを表示するときに必要な操作です。これについては、第10章で説明します。

第10章
ファイル処理

ファイルとファイルデバイス

- ファイルとは
- ファイルデバイスについて
- プログラムファイルの操作
- ファイルの管理

ファイルとファイル名

プログラムや，プログラムの中であつかうデータをひとまとまりにして，コンピューターとコンピューターに接続された機器との間でやりとりすることがあります。

このことを日常生活にたとえてみましょう。あなたが，日記をつけているとします。部屋にはいくつか本だながあって，その中のひとつに「日記」とタイトルのつけられたノートがおさめられています。あなたは，日記を読み返すときも書くときも，まず目的の本だなのところに行って，それから「日記」と書かれたノートをとり出します。

コンピューターの世界に話を戻すと，ノートに相当するのがファイルと呼ばれるものです。(ただし，実際のノートのように手で触れることのできるひとつの固定した”物体”ではなく，必要に応じてカセットテープやフロッピーディスク上に形成される”領域”のようなものと考えてください。)

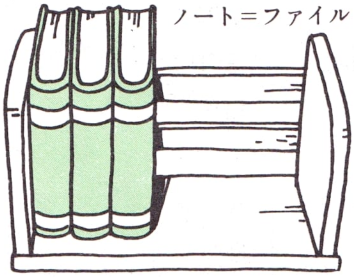
ファイル名

ノートに書かれている「日記」というタイトルは，このノートを他のノートや本と区別するのに役立ちます。同じように，ファイルにも名前をつけることができます。ファイルにつける名前のことをファイル名といいます。

ファイルデバイスとデバイス名

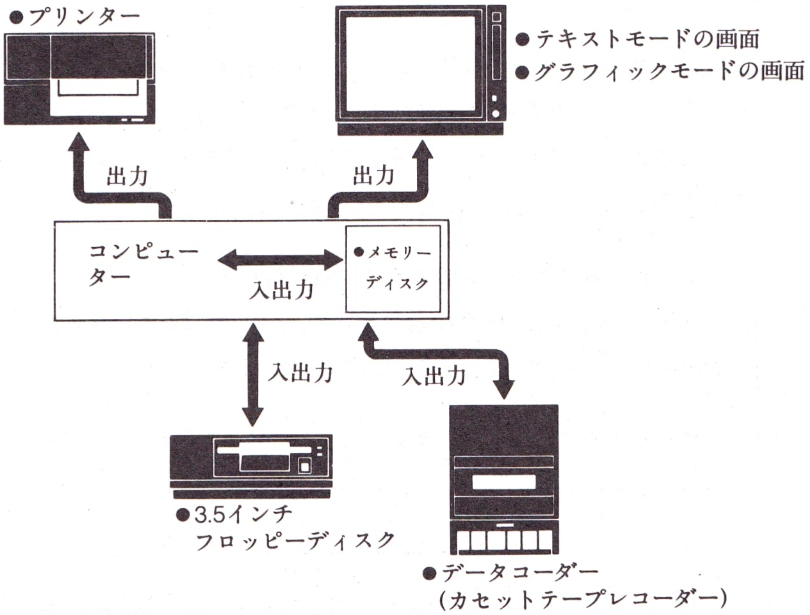
ノートをおさめる本だなは，コンピューターに接続された機器に相当します。接続機器の指定をまちがえると，本だなをまちがえたのと同じように，目的のファイルをとり出すことができません。

MSX-BASICには，コンピューターと特定の接続機器との間でファイルへのやりとりができるように命令語や文が準備されています。ファイルをつくることのできる機器のことをファイルデバイスと呼びます。



ファイルデバイスには，コンピューターからみてプログラムやデータをファイルに出力するだけのものと，入出力ができるものがあります。

MSX_{2+}-BASICで使えるファイルデバイスには次のものがあります。



ファイルデバイス中，フロッピーディスクは，ディスクドライブ内蔵のコンピューターまたは別売りのフロッピーディスクユニットを接続したコンピューターで使うことができます。

デバイス名

各ファイルデバイスのファイルとプログラムやデータのやりとりをする場合，どのファイルデバイスを使うのかを指定しなければならないことがあります。このとき，各デバイスにつけられたデバイス名を使います。

ファイルデバイス	デバイス名
データコーダー(カセットテープレコーダー)	CAS：
テキストモードの画面	CRT：
グラフィックモードの画面	GRP：
プリンター	LPT：
フロッピーディスク	
Aドライブ	A：（ドライブ名）
Bドライブ	B：（ドライブ名）
⋮	⋮
Hドライブ	H：（ドライブ名）
メモリーディスク	MEM：

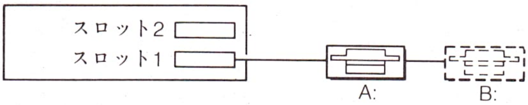
フロッピーディスクの場合、デバイス名の他にドライブ名(A:B:など)を指定しなければならないことがあります。ドライブ名は、ディスクドライブがコンピューターに内蔵されているかどうか、またディスクドライブが外部に接続されている場合はその台数によって変わります。

ディスクドライブのドライブ名について

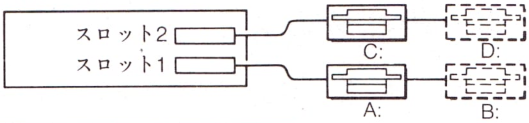
いろいろな組み合わせにおけるドライブ名を図示すると以下のようになります。図中の は仮想ドライブ*を示します。

ディスクドライブのないコンピューター

スロット1または2につないだとき

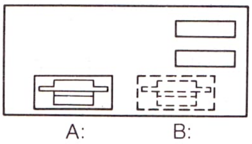


スロット1と2にそれぞれつないだとき

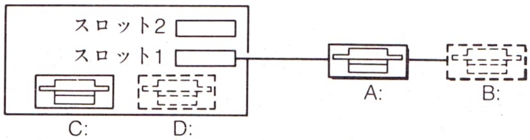


ディスクドライブ1台内蔵のコンピューター

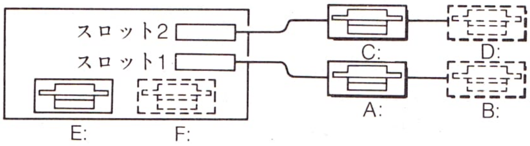
何もつながないとき



スロット1または2につないだとき

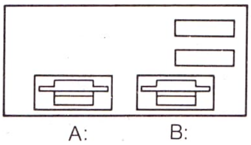


スロット1と2にそれぞれつないだとき

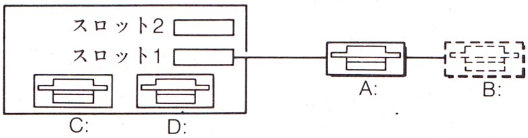


ディスクドライブ2台内蔵のコンピューター

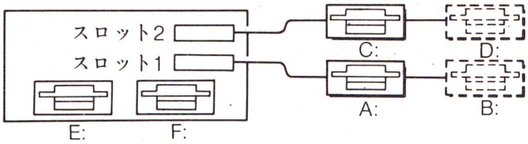
何もつながないとき



スロット1または2につないだとき



スロット1と2にそれぞれつないだとき



***仮想ドライブ**

ディスクドライブが1台でも、これを仮想的にA:B:2台のドライブがあるものとして使うことができます。この実際に存在はしないが、あるものとして使っているドライブのことを仮想ドライブといいます。

ファイル名のルールとタイプ名

ファイル名は，アルファベットで始まる文字列で，文字数はデバイスがカセットテープの場合，6文字まで，それ以外は8文字までです。

ファイル名は，フロッピーディスクを使うときは省略不可能です。それ以外の場合は省略可能ですが，カセットテープやメモリーディスクのファイルに対して入出力を行う場合は，他のファイルと区別できるよう，ファイル名は付けておくようにします。

タイプ名は，ファイル名のあとにピリオドで区切って付けます。タイプ名は3文字までです。

タイプ名は，8文字のファイル名ではどうしてもたりないときや，ファイルの種類を区別するときなどに使います。一例を次にあげておきます。

例1 TESTPROG1 ｝ファイル名は8文字までなので，どちらもTESTPROG
TESTPROG2 ｝ という名前になってしまい区別できない。

そこで，

TESTPROG. 001 TESTPROG. 002
(ファイル名 タイプ名 ファイル名 タイプ名)

のようにタイプ名を使って区別する。

例2

TEST.BAS	—BASICプログラムであることを示すためにBASというタイプ名を付けた。
TEST.ASC	—アスキー形式でセーブされていることを示すためにASCというタイプ名を付けた。
TEST.DAT	—データファイルであることを示すためにDATというタイプ名を付けた。

特に，例2のように，ファイルの種類を区別するためにタイプ名を付けておくと，後のファイル管理がしやすくなります。

プログラムファイルとデータファイル

ファイルの内容がプログラムであるとき，そのファイルをプログラムファイルと呼びます。また，内容がデータのときはそのファイルをデータファイルと呼びます。
データファイルは，データの入出力（書き込みと読み出し）の方法によって，シーケンシャルファイルとランダムアクセスファイルに分かれます。

- ファイル
  - プログラムファイル
  - データファイル
    - シーケンシャルファイル
    - ランダムアクセスファイル

ここでは，プログラムファイルを操作するための全デバイスと全命令語についてまとめておきます。
データファイルについては、次の項で説明します。

プログラムファイルの操作

BASICのプログラムをファイルへセーブしたりファイルからロードしたり，あるいはメモリー中のプログラムと合併するための命令語には次のようなものがあります。

CSAVE, CLOAD…カセットテープ専用

SAVE, LOAD, MERGE…デバイスを指定できる

プログラムのセーブ，ロードとマージ（合併） SAVE, LOAD, MERGE

CSAVE文，SAVE文でセーブしたプログラムは，同じファイル名を指定してCLOAD文，LOAD文によりロードすることができます。このとき，CSAVE文でセーブしたプログラムはCLOAD文でしかロードできず，SAVE文でセーブしたものはLOAD文でしかロードできません。

MERGE文もCLOAD文やLOAD文と同じようにプログラムをメモリーにロードしますが，CLOAD文，LOAD文を実行するとそれまでメモリー上にプログラムがあっても消えてしまうのに対して，MERGE文はメモリー上のプログラムにロードしたプログラムを合併させます。ただし，MERGE文で合併することができるのは，アスキー形式でセーブされたプログラムだけです。

次の表をみてください。

セーブするときの命令語	セーブの形式	ロードするときの命令語
CSAVE	中間言語形式	CLOAD
SAVE（カセットテープ，メモリーディスクの場合）	アスキー形式	LOADまたはMERGE
SAVE（フロッピーディスクの場合）	中間言語形式	LOAD
SAVE, A（フロッピーディスクだけ）	アスキー形式	LOADまたはMERGE

カセットテープの場合は，中間言語形式かアスキー形式かはCSAVE文にするかSAVE文にするかによって決まります。フロッピーディスクの場合はSAVE文しか使えず，Aオプションをつけるとアスキー形式で，つけないと中間言語形式でセーブされます。メモリーディスクの場合もSAVE文しか使えませんが，つねにアスキー形式でセーブされます。

カセットテープでのプログラムファイル操作 CSAVE, CLOAD

メモリー中のプログラムをカセットテープにセーブする例を示します。

```
CSAVE "PROG"          ファイル名 PROG，中間言語形式でセーブ―①
SAVE "CAS:PROG"       ファイル名 PROG，アスキー形式でセーブ―②
```

①でセーブしたプログラムをロードするときは

```
CLOAD "PROG"
```

を実行します。
②でセーブしたプログラムをロードするときは

```
LOAD "CAS:PROG"
```

を実行します。
メモリー中のプログラムに，②でセーブしたプログラムを合併させるときはLOADのかわりに，

```
MERGE "CAS:PROG"
```

を実行します。

フロッピーディスクでのプログラムファイル操作 SAVE. LOAD

メモリー中のプログラムをフロッピーディスク（Aドライブ）にセーブする例を示します。

```
SAVE "A:PROG.BAS"   ―――ファイル／タイプ名 PROG. BASで，
                        中間言語形式でセーブ―①
SAVE "A:PROG.ASC",A ―ファイル／タイプ名 PROG. ASCで，
                        アスキー形式でセーブ―②
```

①，②でセーブしたプログラムをロードするときは，それぞれ

```
LOAD "A:PROG.BAS"
LOAD "A:PROG.ASC"
```

を実行します。

メモリー中のプログラムに，②でセーブしたプログラムを合併させるときはLOADのかわりに，

```
MERGE "A:PROG.ASC"
```

を実行します。

なお，ディスクドライブが1台のときは，ドライブ名（A：）は省略できます。2台以上のときにドライブ名を省略すると，現在使用中のドライブが選択されます。

メモリーディスクを使ったプログラムファイル操作

MSX$_{2+}$-BASICでは，内部メモリーの一部分を，通常プログラムを記憶しておく部分とは別に，プログラムを一時的にセーブしておく場所として使用することができます。あたかも，メモリーをフロッピーディスクのように使用するので，メモリーディスク機能と呼ばれます。

これに対して，通常プログラムを記憶しておくメモリーの部分をプログラムエリアと呼びます。プログラムエリア中のプログラムは，LIST命令で画面に表示したり，PUN命令で実行できます。

プログラムエリア中のプログラムをメモリーディスクにセーブすると，プログラムエリア中には別のプログラムを書き込むことができます。

ただし，電源を切ると，メモリーディスク中のプログラムも消えてしまいます。

メモリーディスクを使う場合は，必ずCALL MEMINI文により，メモリーディスクを初期化しておかなければなりません。

コール　　メモリーイニシャライズ CALL　　MEMINI　　[(サイズ)]

CALL MEMINI文では，どのくらいの領域をメモリーディスクとして使用するかを，サイズで指定できます。サイズはバイト単位で，1023以上32767以下とします。サイズの指定を省略すると，32767バイトとなり，CALL MEMINI文を実行すると次のような表示が出てメモリーディスクの未使用部分の大きさが示されます。

```
CALL MEMINI
32000 bytes allocated
Ok
```

CALL MEMINI文実行後は，次のようにしてプログラムエリアのプログラムをメモリーディスクにセーブすることができます。

```
SAVE "MEM:PROG.BAS"
```
―― ファイル／タイプ名 PROG. BAS, アスキー形式でセーブ

メモリーディスクからプログラムをプログラムエリアにロードするときは，

```
LOAD "MEM:PROG.BAS"
```

を実行します。

プログラムエリア中のプログラムにメモリーディスクのプログラムを合併させるときは

```
MERGE "MEM:PROG.BAS"
```

を実行します。

ファイルの消去 KILL, CALL MKILL

フロッピーディスク上のファイルを消去するときは，KILL命令を使います。たとえば，" TEST. DAT" というファイルを消去するときは

```
KILL "TEST.DAT"
```

を実行します。

メモリーディスク中のファイルを消去するときはCALL MKILL文を実行します。たとえば，" PROG. BAS" というファイルを消去するときは

```
CALL MKILL ("PROG.BAS")
```

を実行します。

CALL（コール） MKILL（メモリーキル） ("ファイル名 [.タイプ名]")

ファイル名の変更 NAME, CALL MNAME

フロッピーディスク上のファイルのファイル名(タイプ名)を変更するときはNAME命令を使います。

NAME（ネーム） "旧ファイル名 [.旧タイプ名]" AS "新ファイル名 [.新タイプ名]"

メモリーディスク上のファイルのファイル名 (タイプ名) を変更するときは，CALL MNAME文を使います。

CALL（コール） MNAME（メモリィネーム） ("旧ファイル名 [.旧タイプ名]" AS "新ファイル名 [.新タイプ名]")

```
NAME "OLD.BAS" AS "NEW.BAS"

CALL MNAME ("OLD.BAS" AS "NEW.BAS")
```

ロードしてすぐ実行 RUN

フロッピーディスク，メモリーディスク中のファイルを，ロードしてすぐ実行させるときは，RUN命令を次の形で実行します。

RUN ”デバイス名：ファイル名［.タイプ名］”

```
RUN "PROG.BAS"
```
── Aドライブの”PROG. BAS”をロードして実行

```
RUN "MEM:PROG.BAS"
```
── メモリーディスク中の”PROG. BAS”をロードして実行

ファイル管理

ファイル名の表示 FILES, CALL MFILES

フロッピーディスク中のファイル名を表示するときは,

```
FILES
```

を実行します。

特定のファイルがディスク上にあるかを確認したいときは,

```
FILES "PROG.BAS"
```

などのように,ファイル/タイプ名を指定します。

メモリーディスク中のプログラムファイル名を表示するときは,

```
CALL MFILES
```

を実行すます。

コール　メモリーファイルズ CALL　MFILES

CALL MFILESを実行すると,プログラムファイル名が表示されるとともにメモリーディスクの残りのバイト数が表示されます。

オートスタートのプログラムファイル

オートスタートのプログラムとは,電源をオンにしたときや[RESET]ボタンを押したときにディスクから自動的にロードされ,実行されるプログラムのことです。

ディスクにプログラムをセーブするときに,ファイル名,タイプ名を

AUTOEXEC. BAS

としておくと,このプログラムはオートスタートのプログラムになります。

オートスタートのプログラムは,ディスク1枚にひとつだけ作ることができます。

シーケンシャルファイルの操作

- シーケンシャルファイルとは
- シーケンシャルファイルへの書き込み
- シーケンシャルファイルからの読み出し
- データの追加
- グラフィック画面に文字を表示
- 1度に開くことのできるファイルの数

シーケンシャルファイルとランダムアクセスファイル

BASICのプログラムで処理したいデータをデバイスとの間でファイルとしてやりとりする場合，データファイルを使います。たとえば，電話帳プログラムというものを考えたとき，このプログラム自体はプログラムファイルにセーブしますが，電話帳プログラムであつかう，人の名前やその電話番号といったデータをセーブするのがデータファイルです。

データファイルには，データを書き込むときも読み出すときも最初から順に行うシーケンシャルファイルと，ファイルの任意の箇所に書き込み，読み出しができるランダムアクセスファイルとがあります。
シーケンシャルファイルと，ランダムアクセスファイルは，使えるファイルデバイスは，それぞれ次のようになっています。

シーケンシャルファイルを使えるデバイス	カセットテープ，フロッピーディスク，プリンター，テキスト／グラフィックモードの画面，メモリーディスク
ランダムアクセスファイルを使えるデバイス	フロッピーディスク

この項では，シーケンシャルファイルについて説明しますが，デバイスには代表としてAドライブのフロッピーディスクを使います。
シーケンシャルファイルへの入出力には，次のような文を使います。

OPEN	ファイルを開く
PRINT # PRINT # USING	ファイルにデータを書き込む
INPUT # LINE INPUT #	ファイルからデータを読み出す
CLOSE	ファイルを閉じる

シーケンシャルファイルへの書き込み OPEN FOR OUTPUT

シーケンシャルファイルに対してデータを出力するときの手順を要約すると次のようになります。

① OPEN文でファイルを開く。
② PRINT#文でデータをファイルに書き込む。
③ CLOSE文でファイルを閉じる。

ファイルを開く

データを出力するときのOPEN文の形式は次のとおりです。

> OPEN(オープン) "［デバイス名］［ファイル名［.タイプ名］］" FOR OUTPUT AS ［#］ファイル番号

OPEN文を実行すると，指定したデバイスに指定したファイル名でファイルに対してデータを出力する準備が整います。コンピューターは，ファイルに対して入出力するときにはあらかじめデータを一定量ため込んでからまとめて入力したり出力したりする，ということを繰り返します。このときにデータをため込むためにメモリー上に用意される領域をバッファと呼び，MSX_{2+}-BASICではこのようなバッファを最大16まで用意することができます。(フロッピーディスクを使う場合は最大7までです。) OPEN文で指定するファイル番号とは，この16バッファのうちのどれを使うかを指定するもので，初期状態では1だけが指定できます。

データを書き込む

OPEN文でファイルを開いたら，実際にデータを出力するのがPRINT#文です。

PRINT(プリント)#ファイル番号，式［セパレーター式…］

ファイル番号は，OPEN文で指定したのと同じファイル番号を指定します。

１回PRINT#文を実行すると，そのPRINT#文で書き込まれたデータの後に自動的にリターンコード（&H0D）とラインフィードコード（&H0A）が追加されます。データ読み込み時には，この２つのコードがデータの区切りを示す目印になります。データが文字型の場合，１つのPRINT#文で複数のデータを書き込むときは，間に","が入るようにします。

たとえば，

```
PRINT #1,A$;",";B$
```

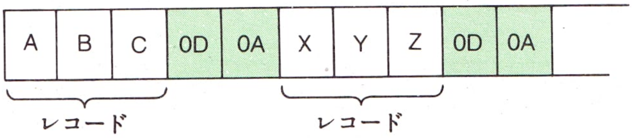
のようにします。カンマも，データの区切りの役割をしますので，この場合も読み込み時にはA$, B$が別々のデータとして扱われます。

データが数値型の場合，ひとつひとつのデータの後に自動的に区切りが入りますから，問題ありません。

たとえば，A$に"ABC"，B$に"XYZ" というデータが代入されている場合，

```
PRINT #1,A$
PRINT #1,B$
```

の2文を実行すると，データは次のように書き込まれます。

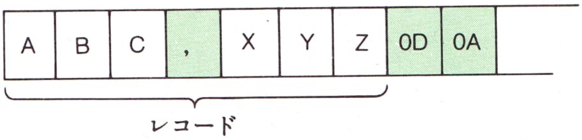


0D, 0Aのペアで区切られたデータのひとまとまりをレコードと呼びます。

また，

```
PRINT #1,A$;",";B$
```

のようにひとつのPRINT#文と，(カンマ)を使うと，データは次のように書き込まれます。



この場合，ひとつのレコード中にABC, XYZが書き込まれますが，間の，(カンマ)により2つのデータとして区別されます。

ファイルを閉じる

データを出力したら，最後にCLOSE文でファイルを閉じます。

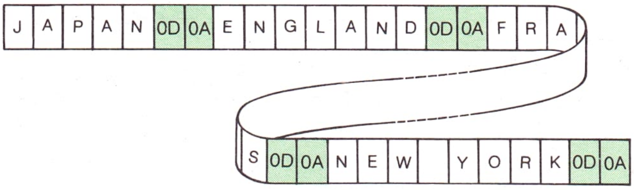
クローズ CLOSE [#] ファイル番号

これで，ファイル番号とファイルとの関係は切り離され，同じファイル番号を使って別のファイルをOPENすることができるようになります。

では，シーケンシャルファイルにデータを書き込むプログラムを作ってみます。

```
10 DIM A$(1,3)
20 OPEN "A:DATA.DAT" FOR OUTPUT AS #1
30 FOR L=0 TO 1
40 FOR M=0 TO 3
50 READ A$(L,M)
60 PRINT #1,A$(L,M)
70 NEXT M
80 NEXT L
90 CLOSE #1
100 END
110 DATA JAPAN,ENGLAND,FRANCE,U.S.A.
120 DATA TOKYO,LONDON,PARIS,NEW YORK
```

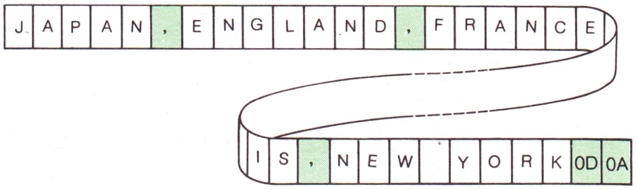
このプログラムを実行すると，Aドライブのディスク上に"DATA. DAT"というファイルができます。そして，このファイルに，まず文字型データJAPAN，次に0D, 0Aのペア，次にENGLANDと，順に書き込まれていきます。



行60を

```
PRINT #1,A$(L,M);",";
```

のようにすると，データは次のように書き込まれることになります。



シーケンシャルファイルからの読み出し　OPEN FOR INPUT

ファイルからデータを入力する手順は次のようになります。

① OPEN文でファイルを開く。

② INPUT#文またはLINE INPUT#文でファイルからデータを読み出す(入力したデータを変数に代入する)。

③ CLOSE文でファイルを閉じる。

ファイルからデータを入力するときのOPEN文の形式は次のとおりです。

```
OPEN "[デバイス名][ファイル名[.タイプ名]]" FOR INPUT AS
 [#] ファイル番号
```

このOPEN文により，ファイルからのデータ入力の準備が整います。ファイル番号は，初期状態ではやはり1だけを指定できます。

ファイルを開いたら，INPUT#文でデータを読み出していきます。

```
INPUT [#] ファイル番号, 変数
```

（INPUT：インプット）

INPUT#文により，ファイルからのデータがひとつ，指定した変数に代入されます。

INPUT#文を使ったときにどのようにデータが読み出されていくかをまとめると次のようになります。

	数値型データの場合	文字型データの場合
データの前のスペース，リターンコード，ラインフィードコード	無視される	無視される
データの区切りとなるもの，あるいはデータが区切られる場合	スペース カンマ リターンコード ラインフィードコード	カンマ リターンコード ラインフィードコード 255字入力したとき
データが"　"で囲まれていた場合	――	""の中をひとつのデータとして入力

なお，LINE INPUT#文は文字データの読み取りだけに使用され，リターンコードだけをデータの区切りとして入力していきます。

データの入力が終わったら，最後にCLOSE文でファイルを閉じ，ファイル番号とファイルの関係を切り離します。

では，先ほど作った"DATA. DAT"というファイルからデータを読み出して画面に表示するプログラムを作ってみます。

```
10 DIM A$(1,3)
20 OPEN "A:DATA.DAT" FOR INPUT AS #1
30 FOR L=0 TO 1
40 FOR M=0 TO 3
50 INPUT #1,A$(L,M)
60 NEXT M
70 NEXT L
80 CLOSE #1
90 FOR M=0 TO 3
100 PRINT A$(0,M),A$(1,M)
110 NEXT M
```

行50のINPUT#文で，配列変数A$にデータを次々と読み出し，行90～110で画面に表示しています。

では，同じ"DATA. DAT"というファイルの内容を次のようなプログラムで読み出そうとするとどうなるでしょうか。

```
10 OPEN "A:DATA.DAT" FOR INPUT AS #1
20 INPUT #1,A$
30 PRINT A$
40 GOTO 20
```

このプログラムを実行すると，文字変数A$に，JAPAN, ENGLAND…と次々と代入されては画面に表示されていきますが，最後のデータNEW YORKを入力してもさらに続けてデータを入力しようとします。このように，ファイルは終わっているのにデータを入力しようとすると，

Input past end

のエラーが起こります。これを防ぐためには，**EOF**関数を用います。

エンドオブファイル **EOF**　（ファイル番号）

```
10 OPEN "A:DATA.DAT" FOR INPUT AS #1
15 IF EOF(1)=-1 THEN GOTO 50
20 INPUT #1,A$
30 PRINT A$
40 GOTO 15
50 CLOSE #1
```

EOF（ファイル番号）という関数は，ファイルの最後のデータが読み出されると−1を返します。上のプログラムでは，この関数を使ってデータをひとつ入力するごとにまだデータがあるかどうかをチェックしているわけです。

データの追加　OPEN FOR APPEND

フロッピーディスクまたはメモリーディスクでシーケンシャルファイルを使っているときだけ，すでに作成したデータファイルにデータを追加していくことができます。その場合，次のようなOPEN文でファイルを開きます。

```
OPEN "［ドライブ名］［ファイル名［.タイプ名］］" FOR APPEND AS ［#］ファイル番号
```

次に，ディスクを使ってデータを書き込むプログラム，データを読み込むプログラム，およびデータを追加するプログラム例を示します。

データの書き込み

```
10 OPEN "A:TEST.DAT" FOR OUTPUT AS #1
20 FOR L=1 TO 3
30 READ A$
40 PRINT #1,A$
50 NEXT L
60 CLOSE #1
70 END
80 DATA JAPAN,ENGLAND,FRANCE
```

このプログラムでは，Aドライブのディスクに"TEST. DAT"というファイルを作って，JAPAN, ENGLAND, FRANCEという3つのデータを書き込んでいます。

データの読み出し

```
10 OPEN "A:TEST.DAT" FOR INPUT AS #1
20 IF EOF(1)=-1 GOTO 60
30 INPUT #1,A$
40 PRINT A$
50 GOTO 20
60 CLOSE #1
70 END
```

データの書き込みのプログラムで書き込んだ3つのデータを読み込んでは変数A$に入れ，画面に表示させています。

データの追加

```
10 OPEN "A:TEST.DAT" FOR APPEND AS #1
20 FOR L=1 TO 2
30 READ A$
40 PRINT #1,A$
50 NEXT L
60 CLOSE #1
70 END
80 DATA U.S.A.,CHINA
```

このプログラムでは，行10で開くファイル”TEST.DAT”には，すでにデータの書き込みプログラムにより3つのデータが書き込まれていますが，FOR APPENDと指定してあるので，行40のPRINT#文で書き込まれるデータは，この3つのデータの後に続けて書き込まれます。したがって，ファイルのデータは，JAPAN, ENGLAND, FRANCE, U.S.A., CHINAの5つになります。
もし，この行10でFOR APPENDとせずにFOR OUTPUTとすると，このファイルの最初からデータを書き込むことになり，最初の3つのデータはなくなってしまいます。

グラフィック画面に文字を表示する

グラフィックモードの画面には，PRINT文で文字を表示させることはできません。グラフィックモードの画面に文字を表示させるには，グラフィックモードの画面をファイルデバイスとみなして，表示させたい文字をシーケンシャルファイルとして出力する，という方法をとります。

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PRINT #1,"How do you do?"
40 GOTO 40
```

上のプログラムを実行すると，画面がグラフィックモードに切り換わり，”How do you do ?”と表示されます。
表示させる位置を指定するには，直前になにかグラフィック命令をひとつ実行します。すると，その命令で最後に指定された位置が，出力される文字列の先頭の文字を構成する8×6ドットの枠の左上隅になります。

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
25 PRESET (100,50)
30 PRINT #1,"How do you do?"
40 GOTO 40
```

このプログラムでは，行25のPRESET命令で使われた位置（100，50）が，行30で出力される文字列の左上端になります。

1度に開くことのできるファイルの数　MAXFILES

MSX$_{2+}$-BASICの初期状態では，ファイル番号は1だけしか指定できません。つまり，ひとつのプログラム中では同時にひとつのファイルしか開くことができないことになります。同時に2つ以上のファイルを開きたいときは，前もって

マックスファイルズ MAXFILES＝ファイル数

によりその数を指定します。

たとえば

```
MAXFILES=5
```

とすると，ファイル番号0から5までの6つのファイルを同時に開くことができるようになります。指定できる値は0～15です。(ディスクを使う場合は0～6)

なお，ファイル番号0はCSAVE，CLOAD，CLOAD?, LOAD, SAVE専用です。

したがって

```
MAXFILES=0
```

を実行すると，以後CSAVE, CLOAD, CLOAD?, LOAD, SAVEの命令しか使用できなくなります。

MAXFILES＝文は，必ずプログラムの先頭で実行するか，直接モードで実行してください。

ランダムアクセスファイルの操作

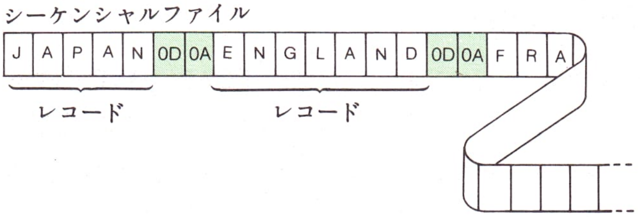
- ランダムアクセスファイルとは
- ランダムアクセスファイルへの書き込み
- ランダムアクセスファイルからの読み出し

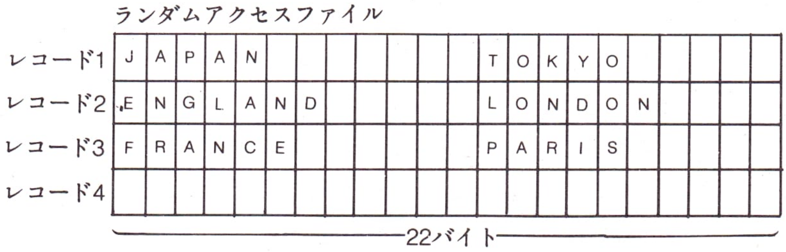
ランダムアクセスファイルとは

ランダムアクセスファイルは，フロッピーディスクだけで使うことのできるファイルです。

シーケンシャルファイルとの比較

シーケンシャルファイルとランダムアクセスファイルのそれぞれにデータが書き込まれているようすを図にすると，次のようになります。





この図からも分かるように，シーケンシャルファイルは，一本の紙テープに，順番にデータを書き込んであるようなものです。1回の読み書きの単位となるレコードの長さは，中のデータの長さによってそれぞれ異なります。

それに対して，ランダムアクセスファイルの方は，一本の紙テープを，あらかじめ同じ長さになるように何本も切っておき，そのひとつを1レコードとしてそれぞれにレコード番号を1，2，3……とつけたもの，と考えることができます。上の図では，紙テープを22字分ずつに切ってあります。

さて，このように，紙テープを何本にも切ったような状態なので，ランダムアクセスファイルでは，このうちのどれかひとつを選んで，そこだけにデータを書き込んだり，そこからデータを読み出したりできるのです。たとえば，レコード3を選ぶと，"FRANCE PARIS"のデータを取り出すことができます。また，いきなりレコード10を指定して，新しいデータを書き込むこともできます。

また，ランダムアクセスファイルでは，それぞれのレコードの中でのデータの書き込まれる位置までが整然と決められています。

ランダムアクセスファイルの操作に使う命令，文

ランダムアクセスファイルへの入出力には，次のような文を使います。

OPEN	……ファイルを開く
FIELD	……1レコード内の変数の割り当てを決める
LSET, RSET	……レコードにデータを書き込む
PUT	……ファイルに1レコード出力
GET	……ファイルから1レコード入力
CLOSE	……ファイルを閉じる

ランダムアクセスファイルへの書き込み

ランダムアクセスファイルに対してデータを出力するときの手順を要約すると次のようになります。

① OPEN文でファイルを開く。
② FIELD文で，1レコード内の変数の割り当てを指定する。
③ LSET文，RSET文で，レコードにデータを書き込む。
④ PUT文でデータを出力する。
⑤ ファイルを閉じる。

これらのステップについて，ひとつひとつに説明を加えます。

① OPEN文でファイルを開く。
ランダムアクセスファイルに対してデータを書き込んだり読み出したりするときもOPEN文でファイルを開きます。そのときのOPEN文の形式は次のとおりです。

```
OPEN ”［デバイス名］ファイル名［.タイプ名］”AS
［#］ファイル番号［LEN＝レコード長］
```

デバイス名は，実際には，ドライブ名を指定します。
これにより，指定したディスクドライブ上に，指定したファイル名，タイプ名でファイルに対してデータを入出力する準備が整います。ランダムアクセスファイルの場合も，1から6までのファイル番号のどれかを指定しますが，初期状態では1だけが指定できます。
また，ランダムアクセスファイルでは，シーケンシャルファイルと異なり，ファイル内の全データの任意の部分を読み出したり書き込んだりできます。このときの読み出し，書き込みの最小単位をレコードと呼びます。OPEN文では，このレコードの大きさを何バイトにするかを指定することができます。指定できる範囲は1から256バイトまでで，指定を省略すると256バイトに設定されます。

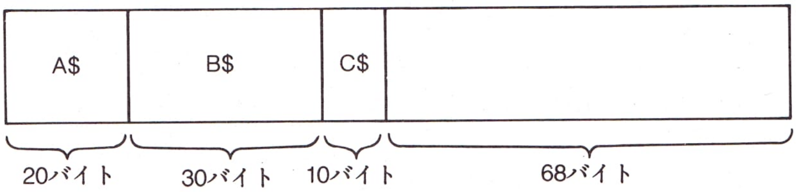
② FIELD文で1レコード内の変数の割り当てを指定する。

FIELD (フィールド) [#] ファイル番号, 文字長 AS 文字変数 [文字長 AS 文字変数] ……

ここでは, どういう変数の値を読み書きするかを宣言するとともに, OPEN文で指定した1レコードをそれぞれの変数のために分割します。ひとつの変数に与えられる長さ（バイト数）が文字長です。
ランダムアクセスファイルで扱えるデータは, すべて文字型データです。

```
FIELD #1,20 AS A$,30 AS B$,10 AS C$
```

この例では, ファイル番号1のファイルに対する入出力用のレコードを分割して, 文字変数A$に20バイト, B$に30バイト, C$に10バイト割り当てています。このとき, 全文字変数の長さの合計は60バイトですから, その前のOPEN文で, レコード長は60バイト以上に設定されていなければなりません。レコード長が128バイトだとすると, 上のFIELD文により1レコードは次のように分割されます。



この場合, 残りの68バイトは何のデータも書き込まれないまま入出力されますので, 書き込むレコードの数が多くなるほどディスク上でむだに使われる部分が増えることになります。したがって, OPEN文でレコード長を60バイトに設定しておく方が良いでしょう。

③　LSET, RSET文で，レコードにデータを書き込む。

FIELD文でレコードを各変数ごとに分割したら，次はファイルに出力したいデータをレコード上にセットします。このための文がLSET, RSETで，LSET文ではデータが左づめで，RSET文では右づめでセットされます。

LSET　文字変数＝文字データ RSET　文字変数＝文字データ

ここで，文字変数はすでにFIELD文でレコード上に割り当てられた変数，文字データは，これらの文字変数を介してファイルに書き込むデータです。

```
LSET A$=X$
```

とすると，レコード上のA$に，文字型データX$が左づめでセットされます。

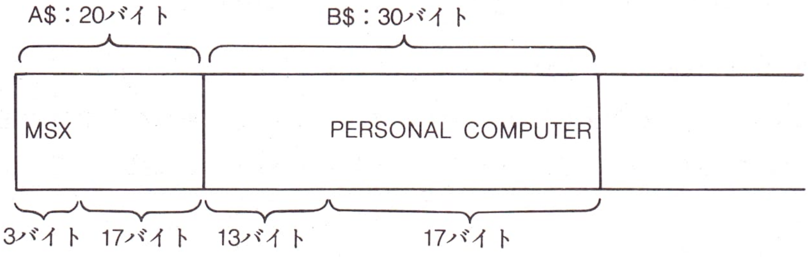
```
RSET B$=Y$
```

とすると，レコード上のB$に，文字型データY$が右づめでセットされます。

たとえば，

```
X$="MSX"
Y$="PERSONAL COMPUTER"
```

となっているとすると，レコード上に次のように書き込まれます。(実際には，入出力用バッファ上に書き込まれることになります。)



もし，書き込んだデータの方がレコード上の変数の長さより長いと，各データの右側の文字がはみ出した分だけ無視されます。

さて，文字データはこれで書き込むことができましたが，レコード上の変数は文字型なので，数値型データはこのままではセットすることはできません。そこで，MKI$, MKS$, MKD$という関数をLSET, RSET文で使って，数値データを文字データに変換してセットします。

LSET（またはRSET）文字型変数＝MKI$(整数型データ)
LSET（またはRSET）文字型変数＝MKS$(単精度型データ)
LSET（またはRSET）文字型変数＝MKD$(倍精度型データ)

LSET P$＝MKI$(X)：LSET Q$＝MKS$(Y)：LSET R$＝MKD$(Z)

このとき，整数型，単精度型，倍精度型のデータを変換して得られる文字データのデータ長はそれぞれ2バイト，4バイト，8バイトと決まっていますから，書き込みたい数値データの型によって，FIELD文で各変数の長さを必要なバイト数に定義しておかなくてはなりません。上の例では，P$は整数型，Q$は単精度型，R$は倍精度型に対応していますから，FIELD文で次のように定義します。

```
FIELD #1,2 AS P$,4 AS Q$,8 AS R$
```

④ PUT文でデータを出力する。

レコード内にデータをセットしたら，PUT文でデータをファイルに書き込みます。

PUT [#] ファイル番号 [,レコード番号]

PUT文により，指定したファイル番号に対応するファイル上の指定したレコード番号の位置に，現在レコードにセットされたデータが書き込まれます。後に，ランダムアクセスファイルからデータを読み出すときは，このレコード番号を指定することになります。

PUT文でレコード番号を省略すると，それまでにPUT文，GET文が実行されていなければ1が指定されたものとみなされます。すでにPUT文，GET文が実行されていれば，その文で使われたレコード番号に1を加えたものが新たなレコード番号として指定されます。

⑤ ファイルを閉じる。

CLOSE#ファイル番号

を実行します。これで，ファイル番号とファイルの関係は切り離されます。

それでは，ランダムアクセスファイルにデータを書き込むプログラム例を示します。

```
10 OPEN "A:TELNO.DAT" AS #1
20 FIELD #1,2 AS ID$,12 AS NAM$,11 AS NO
$
30 FOR R=1 TO 3
40 READ A%,B$,C$
50 LSET ID$=MKI$(A%)
60 LSET NAM$=B$
70 RSET NO$=C$
80 PUT #1,R
90 NEXT R
100 CLOSE #1
110 DATA 1,TOM,111-2222
120 DATA 2,SUSIE,333-4444
130 DATA 3,JOAN,555-6666
```

行40，50のA%という変数は，数値型で整数型の変数です。変数名に%をつけると，整数型の変数になります。

このプログラムを実行すると，Aドライブのディスク上に"TELNO. DAT"というファイル名のファイルが作られます。データの入出力の単位である1レコードの長さは25バイトで，文字変数ID$, NAM$, NO$がそれぞれ2バイト，12バイト，11バイトの領域を持っています。そして，行30からのFOR-NEXTループを3回繰り返すと，ファイルには次のようなデータが書き込まれることになります。

	ID$：2バイト	NAM$：12バイト	NO$：11バイト
レコード1	(1)	TOM	111-2222
レコード2	(2)	SUSIE	333-4444
レコード3	(3)	JOAN	555-6666

ここには，実際には整数型データ1，2，3が内部表現形式に基く文字型データに変換されて各2バイトずつ書き込まれている。

ランダムアクセスファイルからの読み出し

ランダムアクセスファイルからデータを読み出すときの手順も，書き込みの手順とほぼ同じです。

① OPEN文でファイルを開く。

② FIELD文で1レコード内の変数の割り当てを指定する。

③ GET文でデータを入力する。

④ ファイルを閉じる。

ステップの①②は，ファイルへの出力のときと全く同じです。
ステップ③で，GET文を使ってデータを読み出します。

GET [#] ファイル番号 [,レコード番号]

GET文により，指定したファイルから，指定したレコード番号のついたデータの集合，すなわちレコードが読み出され，FIELD文の定義に従って各変数にデータが代入されます。
ところで，数値型のデータは，いったん文字型に変換されてからファイルに書き込まれていますから，読み出したときも文字型のままでFIELD文で定義された文字変数に代入されています。したがって，読み出された数値データ（文字型に変換されている）を画面に表示したり，そのデータを使って計算をするなど何らかの処理を行う場合は，もとの数値型に戻してやらなければなりません。そのために準備されているのが，CVI, CVS, CVDの各関数です。GET文で入力した整数型の数値データが，文字変数P$に代入されたとすると，

```
A%=CVI(P$)
```

によりもとの整数型データに変換されて整数型変数A%に代入されます。画面に表示したいときは，

```
PRINT CVI(P$)
```

のように直接PRINT文中で使うことができます。このように，CVIは整数型に変換する関数ですが，CVSは単精度型に，CVDは倍精度型に変換します。

数値型変数（整数値）=CVI（文字型データ）
数値型変数（単精度型）=CVS（文字型データ）
数値型変数（倍精度型）=CVD（文字型データ）

データの入力が終わったら，やはりCLOSE文でファイルを閉じます。

それでは，先ほどのプログラムで作ったランダムアクセスファイルからデータを読み出して画面に表示するプログラムを作ってみます。

```
10 SCREEN 0:WIDTH 40
20 OPEN "A:TELNO.DAT" AS #1
30 FIELD #1,2 AS ID$,12 AS NAM$,11 AS NO
$
40 INPUT "Record number ";N
50 IF N=0 THEN GOTO 110
60 GET #1,N
70 PRINT "ID NO.";CVI(ID$);"   ";
80 PRINT NAM$;NO$
90 PRINT
100 GOTO 40
110 CLOSE #1
```

プログラムの実行結果は，たとえば次のようになります。

```
RUN
Record number? 1
ID NO.1    TOM                111-2222

Record number? 2
ID NO.2    SUSIE              333-4444

Record number? 0
Ok
```

第11章
機械語サブルーチン

機械語サブルーチンの作成と実行

- 領域の設定と先頭番地の指定—CLEAR, DEFUSR
- 機械語サブルーチンの書き込み—POKE
- 機械語サブルーチンのコール—USR
- 機械語サブルーチンのセーブ，ロード—BSAVE, BLOAD

機械語サブルーチン

MSX-BASICでは，Z-80の機械語を使ったサブルーチンをメモリー内に書き，そのサブルーチンにBASICから制御を移すとともに，機械語サブルーチンの実行結果をBASICで定義した変数に返すことができます。
機械語サブルーチンは最大10個まで作ることができます。また，ひとつの機械語サブルーチンにわたすことのできる値はひとつです。

サブルーチン領域の設定と先頭番地の定義　CLEAR, DEFUSR

機械語サブルーチンを使う場合は，まずCLEAR文によってBASICのプログラムエリアの上限番地を指定し，それ以降のメモリー領域を機械語サブルーチンの書き込み領域とします。

クリア CLEAR［文字領域の大きさ］［，上限番地］

たとえば，

```
CLEAR 300,&HCFFF
```

により，BASICプログラムエリアの上限番地が&HCFFF番地（10進53247番地）に設定されます。したがって，&HD000番地（10進53248番地）以降を機械語サブルーチンの書き込み領域として使用することができるようになります。
なお，このCLEAR文では，文字領域の大きさを300バイトにしています。文字領域の初期値は400バイトあります。

次に，DEFUSR文により，サブルーチンの先頭番地を定義します。

DEFUSR（デフユーザー） [X]＝先頭番地

Xは0から9までの整数です。10個のサブルーチンの先頭番地をUSR関数として定義しておくことができます。

```
DEFUSR0=&HD000
```

このDEFUSR文により，&HD000番地（10進53248番地）からの機械語サブルーチンがUSR0関数として定義されます。

機械語サブルーチンの書き込み　POKE

機械語サブルーチンはPOKE文を使ってメモリーに書き込んでいきます。

POKE（ポーク）　番地，式

POKE文により，メモリーの指定した番地に1バイトのデータを書き込むことができます。

次のプログラムは，メモリーの&HD000番地から，16進数の21，3C，F0，C9というデータを書き込む例です。

```
100 AD=&HD000
110 READ M$:IF M$="END" THEN END
120 POKE AD,VAL("&H"+M$)
130 AD=AD+1:GOTO 110
140 DATA 21,3C,F0,C9
150 DATA END
```

実際には，行140のDATA文中のデータを機械語サブルーチンの命令語（Z-80の命令語）とすれば良いことになります。機械語サブルーチンからは，RET命令によりBASICプログラムに復帰するようにします。

機械語サブルーチンの実行　USR

機械語サブルーチンに制御を移すことを，機械語サブルーチンをコールする，といいます。

機械語サブルーチンをコールするには，USR関数を使います。

変数＝USR（ユーザー） N(K)

Nは，DEFUSR文で定義されたUSR関数の番号，Kは機械語サブルーチンに引き渡す値（数値型または文字型）です。

たとえば，

```
X=USR0(Y)
```

を実行することにより，DEFUSR0により定義された機械語サブルーチンがコールされます。

機械語サブルーチンの実行が終了すると，実行結果の値が変数Xに代入されるとともに，引き続きBASICプログラムが実行されます。

機械語サブルーチンがコールされると，サブルーチンに引き渡された値(変数Yの値)は，メモリー上の次のような位置に格納されるとともに，Yの型によってCPUのAレジスターに型を示すデータが入ります。HLレジスターには，Yの値が格納される領域の先頭番地が入ります。

Yの型	Aレジスターに入るデータ*	HLレジスターの示す番地	Yの値が格納される番地
整数型	2	&HF7F6	&HF7F8-&HF7F9
単精度型	4		&HF7F6-&HF7F9
倍精度型	8		&HF7F6-&HF7FD

＊同じデータがメモリーの&HF663番地にも入ります。

Yが文字型変数の場合は次のようになります。

Aレジスターに入るデータ	DEレジスターに入るデータ	ストリングデスクリプターの内容
3	ストリングデスクリプターの先頭番地	1バイト目：文字列の長さ 2，3バイト目：文字列の格納された領域の先頭番地

機械語サブルーチンの実行が終了すると，終了時にレジスター，メモリーを次のようにしておくことにより，結果の値は変数Xに返されることになります。

結果の値の型	メモリーの&HF663番地	DEレジスター	HLレジスター	結果を格納する番地
整数型	2		&HF7F6	&HF7F8-&HF7F9
単精度型	4		&HF7F6	&HF7F6-&HF7F9
倍精度型	8		&HF7F6	&HF7F6-&HF7FD
文字型	3	ストリングデスクリプターの先頭番地		ストリングデスクリプターの2，3バイト目で示される番地を先頭とする領域

機械語サブルーチンの実例

以上のことを，実際の例でみてみましょう。

まず，機械語プログラムとして，CPUのRレジスター(リフレッシュレジスター)を利用して0～255の乱数を発生させるプログラムを考えておきます。

以下に，そのプログラムのソースリストを示します。(書式は，MACRO80 3.44によります。)

```
        MACRO-80 3.44   09-Dec-81       PAGE    1


                                        .Z80
                                        .PHASE  0D000H

  0002                          INTEGER EQU     2
  F663                          VALTYPE EQU     0F663H
  F7F6                          DAC     EQU     0F7F6H

                                ;
                                ;--     machine-language sample program
                                ;

  D000                          START:
  D000    21 F7F8                       LD      HL,DAC+2
  D003    ED 5F                         LD      A,R             ; load R register
  D005    77                            LD      (HL),A
  D006    23                            INC     HL
  D007    AF                            XOR     A
  D008    77                            LD      (HL),A          ; set random data
  D009    3E 02                         LD      A,INTEGER
  D00B    32 F663                       LD      (VALTYPE),A     ; set data type
  D00E    C9                            RET

                                        END     START
        MACRO-80 3.44   09-Dec-81       PAGE    S


Macros:

Symbols:
F7F6    DAC             0002    INTEGER         D000    START
F663    VALTYPE




No Fatal error(s)


```

発生した乱数は，BASICに返すべき値として，&HF7F8-&HF7F9番地に格納されます。また，返す値の型は整数型なので，&HF663番地には2を入れています。

次は，この機械語サブルーチンをメモリーに書き込み，コールし，サブルーチンより返される値（0〜255の乱数）を利用して，サイコロの1〜6のいずれかを表示するBASICのプログラムです。

```
10 CLEAR 300,&HCFFF ――――BASICプログラム領域の上限番地指定
20 SET BEEP 1
30 '--
40 AD=&HD000:DEFUSR0=AD ――――サブルーチンの先頭番地を&HE000に
50 GOSUB 430
60 ' --
70 SCREEN 5
80 OPEN "GRP:" FOR OUTPUT AS #1
90 SET PAGE 1,1:CLS
100 R=5:RESTORE 360
110 FOR L=0 TO 5
120 XC=(L MOD 3)*80
130 YC=(L ¥ 3)*100
140 LINE (XC,YC)-STEP(50,50),15,BF
150 FOR M=0 TO L
160 READ X,Y
170 CIRCLE (XC+X,YC+Y),R,8
180 PAINT (XC+X,YC+Y),8,8
190 NEXT M
200 NEXT L
    （90〜200：サイコロの目(1〜6)をページ1に描く）
210 ' --
220 SET PAGE 0,0
230 PRESET (70,150)
240 PRINT #1,"Press any key"
250 IF INKEY$="" THEN 250 ――何かキーを押すとサブルーチンをコール
260 J=USR0(0) MOD 20 ――――サブルーチンコール
270 FOR L=0 TO J
280 N=USR0(0) MOD 6 ――――サブルーチンコール
290 X=(N MOD 3)*80
300 Y=(N ¥ 3)*100
310 COPY (X,Y)-STEP(60,60),1 TO (100,70)
,0
    （290〜310：サブルーチンから返された値に従って，サイコロのいずれかをページ0に転送）
320 BEEP:FOR W=0 TO 50:NEXT W
330 NEXT L
340 GOTO 250
```

```
350 ' *** dice data ***
360 DATA 25,25
370 DATA 25,10,25,40
380 DATA 10,10,25,25,40,40
390 DATA 10,10,10,40,40,10,40,40
400 DATA 10,10,10,40,40,10,40,40,25,25
410 DATA 10,10,10,40,40,10,40,40,10,25,4
0,25
420 ' *** write machine language subrout
ine ***
430 RESTORE 480
440 READ M$:IF M$="END" THEN RETURN
450 POKE AD,VAL("&H"+M$)
460 AD=AD+1:GOTO 440
470 ' *** machine codes ***
480 DATA 21,F8,F7,ED,5F,77,23,AF
490 DATA 77,3E,02,32,63,F6,C9
500 DATA END
```

サイコロの目の位置を決めるデータ

機械語サブルーチン書き込み

機械語プログラムのセーブ　BSAVE

今,機械語サブルーチンはメモリーの&HD000-&HD00E番地に書き込まれています。この機械語サブルーチンをカセットテープやディスクにセーブするにはBSAVE命令を使います。

BSAVE（ビーセーブ） "[デバイス名][ファイル名[.タイプ名]]", 先頭番地, 終了番地[,実行開始番地]

BSAVE命令は，メモリーの指定した範囲の内容をセーブします。

```
BSAVE "CAS:RANDOM",&HD000,&HD00E
```

により，&HD000～&HD00E番地の内容（機械語プログラム）をカセットテープにRANDOMというファイル名でセーブすることができます。

```
BSAVE "A:RANDOM.BIN",&HD000,&HD00E
```

により，同じ内容をAドライブのディスクにRANDOM. BINというファイル名，タイプ名でセーブすることができます。

機械語プログラムのロード　BLOAD

BSAVE命令でセーブした内容をロードするときは，BLOAD命令を使います。

BLOAD（ビーロード） "[デバイス名][ファイル名[.タイプ名]]" [,R] [,オフセット]

```
BLOAD "CAS:RANDOM"
```

により,カセットテープからRANDOMというファイル名でセーブされた内容をロードすることができます。

```
BLOAD "A:RANDOM.BIN"
```

によりAドライブのディスクからRANDOM. BINというファイル名,タイプ名でセーブされた内容をロードすることができます。
いずれの場合も,BSAVE命令で指定した先頭番地の位置から始まる領域にロードされます。

このように，機械語サブルーチンだけをセーブしておくと，以後BASICのプログラムでいちいちPOKE文により機械語サブルーチンを書かなくても,ロードするだけですみます。したがって先ほどのBASICのプログラムでは行50および行420～500の部分が不要になります。

さくいん

BASICの命令，文，関数，エラーメッセージなど

A

A： ……43，206
ABS(X) ……170
AND ……104
ASC(X$) ……183
ATN(X) ……169

B

B： ……45，206
BLOAD ……245
BSAVE ……245

C

CALL AKCNV ……180
CALL ANK ……134
CALL CLS ……142
CALL FORMAT ……41
CALL JIS ……180
CALL KACNV ……180
CALL KANJI ……133
CALL KEXT ……180
CALL KINSTR ……179
CALL KLEN ……180
CALL KMID ……179
CALL KNJ ……180
CALL KTYPE ……180
CALL MEMINI ……213
CALL MFILES ……216
CALL MKILL ……214
CALL MNAME ……214
CALL MUSIC ……157
CALL PALETTE ……142
CALL SJIS ……180
CALL VOICE ……160
CAS： ……206
CHR$(X) ……183
CIRCLE ……59
CLEAR ……238
CLOAD ……211
CLOSE ……218，220，229，233
CLS ……28
COLOR ……66，82
COLOR SPRITE ……122
COLOR SPRITE$ ……124
COPY ……45，91，94，99，101
COS(X) ……169
CRT： ……206
CSAVE ……211
CVD ……235
CVI ……235
CVS ……235

D

DATA ……39
DEFUSR ……238
DELETE ……28
DIM ……50
DRAW ……59

E

END ······34
EOF ······223

F

FIELD ······229, 230, 231
FILES ······44, 216
FOR—NEXT ······36

G

GET ······229, 235
GOTO ······29
GRP : ······206

I

IF-THEN ······32
INKEY$ ······187
INPUT ······23
INPUT# ······218, 221
Input past end ······223
INT (X) ······171
INTERVAL OFF ······195
INTERVAL ON ······193
INTERVAL STOP ······199

K

KEY (N) OFF ······195
KEY (N) ON ······193
KEY (N) STOP ······199
KILL ······45, 214

L

LEFT$ (X$,N) ······177
LEN (X$) ······184
LET ······13
LINE ······59
LINE INPUT# ······218
LIST ······20
LOAD ······44, 210, 212
LPT : ······206
LSET ······232

M

MAXFILES= ······227
MEM : ······206
MERGE ······210
MID$ (X$,M,N) ······177
MKD$ ······233
MKI$ ······233
MKS$ ······233
MOD ······68

N

NAME ······214
NEW ······22

O

ON INTERVAL GOSUB ······193
ON KEY GOSUB ······193
ON SPRITE GOSUB ······193
ON STOP GOSUB ······193
ON STRIG GOSUB ······193
OPEN ······218, 221, 224, 230
OR ······102

P

PAINT ······59
PLAY ······144, 162
POKE ······239
PRESET ······59, 103
PRINT ······12, 16
PRINT# ······218, 219
PSET ······59, 103
PUT ······229, 233
PUT SPRITE ······118

R

READ-DATA ······39
REM ······84
RENUM ······35
RIGHT$(X$, N) ······177
RND(X) ······171
RSET ······232
RUN ······21, 215

S

SAVE ······43, 210, 212
SCREEN ······54, 107, 113
SET PAGE ······88
SIN(X) ······169
SOUND ······152
SPACE$(N) ······176
SPC(N) ······176
SPRITE OFF ······195
SPRITE ON ······193
SPRITE STOP ······199
SPRITE$ ······114
SQR(X) ······167
STEP ······62
STICK(N) ······189
STOP OFF ······195
STOP ON ······193
STOP STOP ······199
STR$(X) ······181
STRIG(N) OFF ······195
STRIG(N) ON ······193
STRIG(N) STOP ······199
Syntax error ······18

T

TAN(X) ······169
TIME ······174
Type mismatch ······18

V

VAL(X$) ······181

W

WIDTH ······57

X

XOR ······103

用語など

あ

アクティブページ……86
アスキー形式……210
色バケ……70，80，81，82
インターレース走査……109
インターレースモード……109
英数字入力モード……132
FM音源……157
エラーメッセージ……18
オートスタートのプログラム……216

か

カーソルキー……26，189
かな漢字変換モード……135
カラーコード……63
カラーパレット……64
関数……166
カンマ（,）……27
画面のモード……54
機械語サブルーチン……238
機械語サブルーチンのコール……240
キャラクターモード……57
キークリックスイッチ……107
キーボード……11
輝度……65
機能強化スプライト……122
行番号……20
グラフィックモード……59
効果音……152
CTRL＋STOP……30

さ

座標……59
シーケンシャルファイル……209，217
シングルクォート……84
条件式……33，34
条件判断……33
数値型・文字型変換関数……181
数値型の関数……166
数値型の変数……17
スプライト……112
スプライトの大きさ……113
スプライトパターン……112，113
スプライト面……56
スペース……17
セーブ……43
セミコロン（；）……26

た

タイプ名……208
代入……39
チャンネル（FM音源の）……157
中間言語形式……210
直接モード……19
転送……90
データ……40，209
データ入力関数……185
データファイル……209
ディスプレイページ……86
デバイス名……206
透明色……63
ドライブ名……206

な

日本語入力モード……132
ヌルストリング……187
ノンインターレース走査……109

は

配列変数……50
80字モード……57
バックグラウンド……54, 56
パレット機能……63, 64, 65
PSG(ピーエスジー)……144
ファイル……204
ファイルデバイス……205
ファイル名……44, 204
フォーマット……41
フォアグラウンド……54, 56
FOR-NEXT(フォーネクスト)ループ……37
フローチャート……32
文……33
プリンタータイプ……107, 108
プログラム……20, 21
プログラムエリア……213
プログラムの修正……25
プログラムファイル……209
プログラムモード……19
プログラムリスト……20
プロンプト文（INPUT文の）……24
変数……13
ページ……85
ページ切り換え……88
ページ番号……86
ボーダーエリア……54, 56
ボーレート……107, 108

ま

マージ……210
マルチカラーモード……60
無限ループ……30
メモリーディスク機能……213
文字型の関数……176
文字型変数……17
文字コード……132, 183
文字の足し算……16
文字列……16

や

40字モード……57

ら

ラジアン……169
乱数……171
ランダムアクセスファイル 209, 217
リズム音……159
RESET(リセット)ボタン……22
ループ……30
レコード……219, 228, 229
レジスター（PSGの）……152
連文節変換……137
ロード……44
ロジカルオペレーション……101

わ

YJK(ワイジエーケー)方式……71
割り込み……192
割り込み宣言文……193

こんなときには

操作

BASICをスタートさせる……………11
コンピューターに命令する ………12
ディスクのフォーマット …………41
オートスタートの
プログラムを作る …………………216

プログラムの作成

プログラムの入力 …………………19
プログラムのリストを
表示（LIST） ………………………20
プログラムを消去（NEW） ………22
プログラムの行を追加 ……………25
特定の行を表示（LIST） …………26
プログラムの行を修正 ……………26
プログラムの行を
消す（DELETE） ……………………28
行番号を整理する（RENUM） ……35

プログラムの実行と流れの制御

プログラムを実行する（RUN） …21
プログラムを終了させる（END）…34
ジャンプする（GOTO） …………29
無限ループを作る（GOTO） ……29
ループを作る（FOR-NEXT） ……36
ループの中にループを作る ………38
条件判断をする（IF-THEN-ELSE） 32

データの代入，表示

変数に数値を代入（LET） …………13
計算結果を表示する（PRINT 式）…12
LETの省略 …………………………15
文字列を表示（PRINT〝文字列〞）…16
文字列の足し算 ……………………16
キーボードから変数に値を
代入（INPUT） ……………………23
INPUT文でプロンプト文を使う …24
変数に代入する値を
ひとまとめに（READ-DATA）……39

数値データの処理

計算する（PRINT 式） ……………12
数値型を文字型に
変換（STR$(X)） …………181，182
平方根を求める（SQR(X)） ……167
三角関数を使う ……………………169
絶対値を求める（ABS(X)） ……170
乱数を得る（RND(X)） …………171
整数に直す（INT(X)） …………171
文字コードに対応する文字を
得る（CHR$(X)） …………………183

文字型データの処理

文字列を表示（PRINT〝文字列〞）…16
文字列の足し算 ……………………16
文字列の左の一部分を
得る（LEFT$(X$, N)） ……………177
文字列の右の一部分を
得る（RIGHT$(X$, N)） …………177
文字列の一部分を
得る（MID$(X$, M, N)） …………177
文字コードを得る（ASC(X$)） …183
文字列の長さを得る（LEN(X$)）…184

プログラムのセーブとロード

ディスクにセーブ（SAVE）………43
ディスクからロード（LOAD）……44
プログラムの合併（MERGE）…210
メモリーディスクに
セーブ（SAVE）…………………213
メモリーディスクから
ロード（LOAD）…………………213
ロードしてすぐ実行（RUN）……215

データファイル

シーケンシャルファイルへの
書き込み…………………………218
シーケンシャルファイルからの
読み出し…………………………221
シーケンシャルファイルにデータを
追加………………………………224
ランダムアクセスファイルへの
書き込み…………………………230
ランダムアクセスファイルからの
読み出し…………………………235

ファイル管理

ディスク上のプログラムの
コピー（COPY）…………………45
ディスク上の
ファイル表示（FILES）……44，216
ディスク上の
ファイルの消去（KILL）……45，214
メモリーディスク上のファイルの
表示（CALL MFILES）…………216
メモリーディスク上のファイルの
消去（CALL MKILL）……………214
ファイル名の変更
（NAME，CALL MNAME）……214
1度に開くことのできるファイル数の
指定（MAXFILES）………………227

定義および設定

画面のモードを指定（SCREEN）…54
配列変数を宣言する（DIM）……50
メモリーディスクの
初期化（CALL MEMINI）………213
カセットボーレートの
設定（SCREEN）…………………108
プリンタータイプの
指定（SCREEN）…………………108
日本語入力モードにする（CALL
KANJI）……………………………133
FM音源で演奏できるようにする(CALL
MUSIC)……………………………157

キャラクターモードで

1行の文字数を
指定する（WIDTH）………………57
画面の表示を消す（CLS）………28

グラフィックモードで

インターレース走査を
する（SCREEN）…………………109
画面の表示を消す（CLS）………28
ページを切り換える

(SET PAGE) ……………………88
図形データを転送する
(COPY) …………91, 94, 99, 101
ロジカルオペレーションを使って
転送する ……………………………105

日本語入力モードで

日本語文字列の一部分を得る (CALL KMID) ……………………………179
日本語文字列の検索 (CALL KINSTR) ………………………………………179
日本語文字列の長さを得る (CALL KLEN) ……………………………180
日本語の文字コードを得る(CALL JIS, CALL SJIS) ………………………180
日本語文字コードから文字を得る (CALL KNJ) ……………………180
全角, 半角変換(CALL AKCNV, CALL KACNV) …………………………180
全角, 半角分離 (CALL KEXT) 180
全角, 半角検出 (CALL KTYPE) 180
英数字入力モードに戻す(CALL ANK) ………………………………………134
画面の表示を消す (CALL CLS) 142

カラー

カラーパレットを
設定する (COLOR) ………………66
SCREEN 8モードのカラーコードを求める ………………………………69

スプライト

スプライトパターンを定義する …114
スプライトパターンを
表示する (PUT SPRITE) ………118
スプライトを動かす ………………120
スプライトの色を
変える (COLOR SPRITE) ……122
スプライトの1ラインごとに色を
指定する (COLOR SPRITE$) …124
スプライトの1ラインを32ドット
ずらす ………………………………126

音楽の演奏

PSGでメロディーを演奏 (PLAY) ………………………………………144
効果音を作る (SOUND) ………152
FM音源で演奏できるようにする(CALL MUSIC) ……………………………157
FM音源の音色を決める(CALL VOICE) ………………………………………160
FM音源で演奏する (PLAY) ……162

プログラミングのテクニック

スプライトパターン定義の
テクニック …………………………128
実行ごとに異なる乱数を得る ……174
キーを押すことによってプログラムを
進める ………………………………187
特定のキーを押すことによって
プログラムを進める ………………189
グラフィック画面に文字を
表示する ……………………………226

付録
MSX2+ーBASIC
リファレンスチャート

目次

・MSX2+ーBASICコマンド一覧……256
・エラーメッセージ一覧……268
・関数一覧……269
・定数と変数の扱い……270
・カラーコード表……270
・演算記号一覧……270

□：ディスクマーク　　漢：日本語入力モード
▣：メモリーディスクマーク　♪：FM音源（MSX-MUSIC）マーク

コマンド

プログラムの作成に関するコマンド

形　　式	機　　能	例
AUTO 〔先頭行番号〕[,増分]	行番号を自動発生する。	AUTO 100, 10
DELETE [行番号] [-行番号]	プログラム中の行を消去する。	DELETE 30-60
LIST [先頭行番号] [-] [最終行番号]	プログラムリストを表示する。	LIST
NEW	プログラムを消去する。	
RENUM[新先頭行番号],[旧先頭行番号],[増分]	行番号をつけ直す。	RENUM 100, 10, 10
REMまたは	注釈文をつける。	REM…PROGRAM 1…
KEY LIST	ファンクションキーの内容を表示する。	

定義および設定に関するコマンド

形　　式	機　　能	例
CLEAR [文字列領域の大きさ] [,上限番地]	数値変数を0に、文字変数をヌルストリングにし、文字列領域の大きさとBASICが使用するメモリーの上限番地を設定する。	CLEAR 400, 55296
DIM 変数名(添字の最大値[, 添字の最大値]…)	配列の名前、型、大きさ、次元を宣言する。	DIM A$(100)
DEF{INT / SNG / DBL / STR}文字[-文字] [, 文字[-文字]]…	変数名の頭文字と変数の型の対応を定義する。(INT：整数型、SNG：単精度型、DBL：倍精度型、STR：文字型)	DEFINT I-N
DEF FN 関数名[(パラメーター)]=式	ユーザー関数を定義する。	DEF FNA(X)=A＊X ^2+B＊X+C
ERASE 配列変数名[,配列変数名]……	配列を抹消する。	ERASE A,B,C
KEY ファンクションキー番号, 文字列	ファンクションキーに文字列を定義する。	KEY1,“LLIST”+CHR$(13)
SET VIDEO [モード],[輝度],[カラーバス],[同期],[音声],[ビデオ入力],[AVコントロール]	スーパーインポーズその他のモードを設定する。(スーパーインポーズ機能付きのコンピューターでのみ使用可) モード…… 0：コンピューター信号(内部同期のみ) 1：コンピューター信号 2：スーパーインポーズ 3：テレビ信号 輝度…… 0：通常 1：半分 カラーバス…… 0：VDPのカラーバス入力 1：VDPのカラーバス出力 同期…… 0：内部同期 1：外部同期 音声…… 0：外部音声を混合しない 1：右チャンネルを混合 2：左チャンネルを混合 3：両チャンネルを混合 ビデオ入力…… 0：RGBマルチコネクター 1：外部映像入力コネクター AVコントロール…… 0：RGBマルチコネクターのAVコントロール端子出力OFF 1：出力ON	SET VIDEO 2

プログラムの実行およびその流れのコントロールに関するコマンド

形　　　式	機　　　能	例
RUN ［行番号］	実行を開始する。	RUN 100
RUN"［ドライブ名］ファイル名［.タイプ名］"［,R］	ディスク、メモリーディスクのプログラムをロードして実行する。	RUN"PROG. BAS"
STOP	実行を中断する。	
CONT	実行を再開する。	
END	実行を終了する。	
TRON	実行した行番号を表示する。	
TROFF	TRONを解除する。	
FOR 変数＝初期値 TO 最終値 ［STEP増分］ ⋮ NEXT［変数］	FOR-NEXTで囲まれたプログラムの実行を繰り返す。	FOR I=1 TO 10 STEP 2 ⋮ NEXT I
GOSUB 行番号 ⋮ RETURN ［行番号］	指定したサブルーチンへ実行を移す。RETURN文で、メインルーチンに実行が戻る。	100 GOSUB 1000 ⋮ 1000 ⋮ 1100 RETURN
GOTO 行番号	指定した行番号へ実行を移す。	GOTO 100
IF 式 {THEN {文／行番号}／GOTO 行番号} ［ELSE {文／行番号}］	式の値によって実行を分岐する。	IF X=0 THEN 100 ELSE 200
ON 式 GOTO 行番号［,行番号］……	式の値によって実行を分岐する。	ON A GOTO 100,200,300
ON 式 GOSUB 行番号［,行番号］……	式の値によって実行を分岐する。	ON SGN(A)+2 GOSUB 1000,2000,3000

スクリーンに関するコマンド

形　　　式	機　　　能	例
SCREEN ［モード］,［スプライトの大きさ］,［キークリックスイッチ］,［ボーレート］,［プリンタータイプ］,［インターレースモード］	画面表示モードを設定する。 モード…… 0：80×24字テキストモード 1：32×24字テキストモード 2：256×192ドット, 16色グラフィックモード 3：64×48ドット, 16色マルチカラーモード 4：256×192ドット, 16色グラフィックモード, スプライト強化 5：256×212ドット, 16色グラフィックモード, スプライト強化 6：512×212ドット,4色グラフィックモード, スプライト強化 7：512×212ドット,16色グラフィックモード, スプライト強化 8：256×212ドット, 256色グラフィックモード, スプライト強化 10：256×212ドット, 12499色グラフィックモード, スプライト強化 11：256×212ドット, 12499色グラフィックモード, スプライト強化	SCREEN 2,0,0

形式	機能	例
SCREEN [モード],[スプライトの大きさ],[キークリックスイッチ],[ボーレート],[プリンタータイプ],[インターレースモード]	12：256×212ドット 19268色グラフィックモード、スプライト強化 スプライトの大きさ…… 0：8×8ドット標準 1：8×8ドット拡大 2：16×16ドット標準 3：16×16ドット拡大 キークリックスイッチ…… 0：キー音を消す 1：キー音を出す ボーレート…… 1：1200ボー 2：2400ボー プリンタータイプ…… 0：MSXプリンター 1：MSXプリンター以外 インターレースモード…… 0：ノンインターレース 1：インターレース 2：ノンインターレース、偶数／奇数ページ切り換え表示 3：インターレース、偶数／奇数ページ切り換え表示	
SET PAGE[ディスプレイページ],[アクティブページ]	ディスプレイページとアクティブページを指定する。	SET PAGE 0,1
WIDTH 一行表示文字数	テキストモードでの一行の文字数を指定する。	WIDTH 28
CLS	スクリーン上のすべての表示を消去する。	
KEY {ON / OFF}	ファンクションキーの内容表示を出す。または消す。	KEY OFF
LOCATE [X座標],[Y座標],[カーソルスイッチ]	カーソルを移動する。 カーソルスイッチ…… 0：カーソルを表示しない 1：カーソルを表示する	LOCATE 10,12,1
COLOR [フォアグラウンドカラー],[バックグラウンドカラー],[ボーダーカラー]	フォアグラウンドの色、バックの色、ふちどりの色を指定する。	COLOR 8,15,2
COLOR=(パレット番号，赤の輝度，緑の輝度，青の輝度)	カラーパレットに色を割り当てる。	COLOR=(2,0,3,7)
COLOR=RESTORE	ビデオRAM内のカラールックアップテーブルの内容をVDPのカラーパレットレジスターに代入する。	
COLOR [=NEW]	カラーパレットの内容を初期状態に戻す。	
PUT SPRITE スプライト面番号，[[STEP](X座標,Y座標)],[カラー]，[スプライト番号]	指定したスプライトを指定したスプライト面の指定位置に表示する。	PUT SPRITE 0,(100,50),7,2
COLOR SPRITE$(スプライト面番号)="文字式"	スプライトの各ラインごとに色を指定する。文字の各ビットの働き： B7 B6 B5 B4 B3 B2 B1 B0 B7 ：1のとき、そのラインを32ドット左へずらす。	COLOR SPRITE$(0)=CHR$(1)+CHR$(7)

形　　式	機　　能	例
COLOR SPRITE$(スプライト面番号)＝"文字式"	B6　：1のとき，スプライトの優先順位と衝突を無視し，スプライトの重なりの部分はカラーコードのORをとる。 B5　：1のとき，スプライトの衝突を無視する。 B4　：使用されない B3～B0：カラーコード	
COLOR SPRITE(スプライト面番号)＝パレット番号	指定したスプライト面上のスプライトの色を変える。	COLOR SPRITE(1)＝4
SET SCROLL [X] [,Y] [,マスク] [,ページ]	ディスプレイページをスクロールする。	SET SCROLL 128, 100, 0, 0

ロジカルオペレーション
PSET, PRESET, AND, OR, XOR
TPSET, TPRESET, TAND, TOR, TXOR

グラフィック表示に関するコマンド

形　　式	機　　能	例
CIRCLE [STEP](X座標, Y座標), 半径, [カラー], [開始角], [終了角], [縦横比]	円を描く。	CIRCLE (80,60),15,8
DRAW "サブコマンド"	任意の図形を描く。	DRAW "S40U5R5D5L5"
LINE [[STEP](X座標,Y座標)]−[STEP](X座標, Y座標), [カラー] {,[B] / ,[BF]}, [ロジカルオペレーション]	直線または四角を描く。	LINE-STEP(20,50),,B
PAINT [STEP](X座標,Y座標),[表示カラー],[境界線カラー]	境界線で囲まれた領域を塗りつぶす。	PAINT (120,100)
PSET [STEP](X座標,Y座標),[カラー],[ロジカルオペレーション]	点を打つ。	PSET STEP(10,10),14
PRESET [STEP](X座標,Y座標),[カラー],[ロジカルオペレーション]	点を打つ。または消す。	PRESET (100,100)
PUT KANJI [(X座標,Y座標)],漢字コード,[カラー],[ロジカルオペレーション],[モード]	漢字を指定位置に表示する。 モード…… 0：16×16ドットで表示 1：偶数番目のラインのみ たて8ドットで表示 2：奇数番目のラインのみ たて8ドットで表示	PUT KANJI (50,50),&H3441,10

画面データの処理に関するコマンド

形　　式	機　　能	例
COPY (X1,Y1)−(X2,Y2)[,転送元ページ]TO (X3,Y3),[転送先ページ],[ロジカルオペレーション]	ビデオRAM内の画面データをビデオRAM内の他の領域に転送する。	COPY(20,30)−(70,50),1 TO(90,60),0,AND
COPY (X1,Y1)−(X2,Y2)[,転送元ページ] TO 配列変数名	ビデオRAM内の画面データを配列変数に転送する。	COPY (20,30)−(70,50),0 TO S
COPY 配列変数名[,向き] TO (X3,Y3),[転送先ページ],[ロジカルオペレーション]	配列変数内の画面データをビデオRAMに転送する。	COPY S,1 TO (100,100),1,XOR
💾COPY (X1,Y1)−(X2,Y2) [,転送元ページ]TO "[ドライブ名：]ファイル名[.タイプ名]"	ビデオRAM内の画面データをディスクのファイルにセーブする。	COPY (10,10)−(120,90) TO"PORTRAIT.PIC"
💾COPY "[ドライブ名]ファイル名[.タイプ名]"[,向き]TO(X3,Y3), [転送先ページ], [ロジカルオレペーション]	ディスクのファイル内の画面データをビデオRAMにロードする。	COPY "PORTRAIT.PIC" TO (10,10)

形式	機能	例
💾COPY "[ドライブ名] ファイル名 [.タイプ名]" TO配列変数名	ディスクのファイル内の画面データを配列変数にロードする。	COPY "PORTRAIT.PIC" TO S
💾COPY 配列変数 TO "[ドライブ名] ファイル名 [.タイプ名]"	配列変数内の画面データをディスクのファイルにセーブする。	COPY S TO "PORTRAIT.PIC"
COPY SCREEN [モード], [マスク]	外部映像信号をデジタイズしてVDPに書き込む。 (デジタイズ機能つきのコンピューターでのみ使用可) モード…… 0：1フィールドの信号をデジタイズしてディスプレイページに書き込む。 1：2フィールド(1フレーム)の信号をデジタイズしてディスプレイページと、それよりページ番号が1小さいページに書き込む	

グラフィックサブコマンド (Bをつけると線を描かず始点だけ動かす。Nをつけると線は描くが始点は移動しない)

サブコマンド	機能	初期値	サブコマンド	機能	初期値
Mx, y	絶対位置x, yへ		Fn	右下へ	n=1
M±x, ±y	現在位置からx, yだけ移動		Gn	左下へ	n=1
Un	上へ	n=1	Hn	左上へ	n=1
Dn	下へ	n=1	An	座標系を回転する。	
Rn	右へ	n=1	Cn	色を設定する。	n=15
Ln	左へ	n=1	Sn	1単位のドット数を指定する。	n=4
En	右上へ	n=1	X文字型変数;	文字型変数に代入されたサブコマンドを実行。	

サウンドに関するコマンド

形式	機能	例
BEEP	ピッという音を鳴らす。	BEEP：BEEP：BEEP
SOUND PSGのレジスター番号, 式	PSGのレジスターにデータを書き込む。	SOUND 7,7
PLAY [#モード,] サブコマンド1 [, サブコマンド2] … [, サブコマンド12]	音楽を演奏する。	PLAY "O4L4CEGEL1C"
♪ CALL MUSIC [([モード] [, [0] [,PLAY文第1サブコマンド列用のチャンネル数 [, 第2サブコマンド列用のチャンネル数 [, … [,第9サブコマンド列用のチャンネル数]]]]]]]]])]	FM音源を使用可能にする	CALL MUSIC(0, 0, 1, 2, 1)
♪ CALL AUDREG(レジスター番号, 式[, チャンネル番号])	FM音源LSIのレジスターにデータを書き込む。	CALL AUDREG(&H20,0)
♪ CALL BGM(N)	バックグラウンドで演奏するかフォアグラウンドで演奏するかを指定する。	CALL BGM(0)
♪ CALL PITCH(ピッチ)	FM音源のメロディー音の高さを指定する。	CALL PITCH(450)
♪ CALL STOPM	バックグラウンドで実行中のPLAY文による演奏を停止する。	
♪ CALL TEMPER(音律)	FM音源の音律を指定する。	CALL TEMPER(0)
♪ CALL TRANSPOSE(トランスポーズ値)	FM音源のメロディー音の高さをセント単位で移調する。	CALL TRANSPOSE(600)
♪ CALL VOICE([チャンネル1用の音色],[チャンネル2用の音色],…,[チャンネル9用の音色])	FM音源の音色を設定する。	CALL VOICE(@0,@2,@7)

形式	機能	例
♪ CALL VOICE COPY(音色1, 音色2)	音色のデータを転送する。	CALL VOICE COPY(@17, @63)
♪ CALL PLAY(PLAY文のサブコマンド列番号, 変数)	指定したサブコマンド列が演奏中かを調べる。	CALL PLAY(2, A)

ミュージックサブコマンド

サブコマンド	機能と設定範囲	初期値	サブコマンド	機能と設定範囲	初期値
A[#/+/−]−G[#/+/−]	音符		Tn	テンポ 32≦n≦255	n=120
On	オクターブ 1≦n≦8	n=4	Vn	音量 0≦n≦15	n=8
Nn	音程 0≦n≦96		Mn	エンベロープ周波数 1≦n≦65535	n=255
Ln	長さ 1≦n≦64	n=4	Sn	エンベロープパターン 1≦n≦15	n=1
Rn	休符 1≦n≦64	n=4	・	符点	
X文字型変数;	文字型変数に代入されたサブコマンドを実行		&	タイ	
♪ { } n	連符 1≦n≦64	Lnでの設定値	♪ @n	n番の音色に切り換える 0≦n≦63	
♪ @Vn	音量を細かく設定 0≦n≦127		@Wn	nの長さだけ状態を継続	Lnでの設定値
♪ Yr, d	FM音源LSIのレジスターrにデータdを書き込む		>	オクターブをひとつ上げる	
<	オクターブをひとつ下げる		♪ Qn	音の長さの割合 1≦n≦8	Q8

♪ リズム音のサブコマンド

サブコマンド	機能と設定範囲	初期値	サブコマンド	機能と設定範囲	初期値
B	バスドラム音を発生		n	直前までの楽器音を発生しn分音符分待つ 1≦n≦64	
S	スネアドラム音を発生		Vn	アクセントなしの楽器音の音量 0≦n≦15	8
M	タムタム音を発生		@An	アクセントつきの楽器音の音量 0≦n≦15	
C	シンバル音を発生		Tn	テンポを決める 0≦n≦255	120
H	ハイハット音を発生		@Vn	音量を細かく設定する 0≦n≦127	
!	直前の楽器にアクセント		Rn	休符を指定する 1≦n≦64	R4

♪ FM音楽の音律

音律の指定	設定される音律	音律の指定	設定される音律
0	ピタゴラス	11	純正律 cisメジャー(bマイナー)
1	ミーントーン	12	純正律 dメジャー(hマイナー)
2	ヴェルクマイスター	13	純正律 esメジャー(cマイナー)
3	ヴェルクマイスター（修正1)	14	純正律 eメジャー（cisマイナー)
4	ヴェルクマイスター（修正2)	15	純正律 fメジャー（dマイナー)
5	キルンベルガー	16	純正律 fisメジャー（esマイナー)
6	キルンベルガー（修正)	17	純正律 gメジャー（eマイナー)
7	ヴァロッティ・ヤング	18	純正律 gisメジャー（fマイナー)
8	ラモー	19	純正律 aメジャー（fisマイナー)
9	完全平均律（初期設定)	20	純正律 bメジャー(gマイナー)
10	純正律 cメジャー（aマイナー)	21	純正律 hメジャー（gisマイナー)

♪ FM音源の音色

音色番号	音色	音色番号	音色
@0	ピアノ1＊	@32	ピアノ3
@1	ピアノ2	@33	エレクトリックピアノ2＊
@2	バイオリン＊	@34	サンツール2
@3	フルート1＊	@35	ブラス
@4	クラリネット＊	@36	フルート2
@5	オーボエ＊	@37	クラビコード2
@6	トランペット＊	@38	クラビコード3
@7	パイプオルガン1	@39	琴2
@8	シロフォン（木琴）	@40	パイプオルガン2
@9	オルガン＊	@41	PohdsPLA
@10	ギター＊	@42	PohdsPRA
@11	サンツール1	@43	チャーチオルガンL
@12	エレクトリックピアノ1＊	@44	チャーチオルガンR
@13	クラビコード1	@45	シンセ・バイオリン
@14	ハープシコード1＊	@46	シンセ・オルガン
@15	ハープシコード2	@47	シンサ・ブラス
@16	ビブラホン＊	@48	ホルン＊
@17	琴1	@49	三味線
@18	太鼓	@50	マジカル
@19	エンジン1	@51	フワワ
@20	UFO	@52	ワンダーフラット
@21	シンセサイザーベル	@53	ハードロック
@22	チャイム	@54	マシン
@23	シンセ・ベース＊	@55	マシンV
@24	シンセサイザー＊	@56	コミック
@25	シンセ・パーカッション	@57	SE-コミック
@26	シンセ・リズム	@58	SE-レーザー
@27	ハーモ・ドラム	@59	SE-ノイズ
@28	カウベル	@60	SE-星1
@29	ハイハット	@61	SE-星2
@30	スネア・ドラム	@62	エンジン2
@31	バス・ドラム	@63	無音

＊のついていない音色どうしを同時に2つ以上設定することは不可。

プログラムおよびデータのファイルに関するコマンド

形　　　　式	機　　　　能	例
MAXFILES＝式	1回で開くことのできるファイル数を設定する。	MAXFILES＝3
OPEN "デバイス名 [ファイル名] [.タイプ名]" [FORモード] AS [#] ファイル番号 [LEN＝レコード長]	ファイルを開く。 モード……OUTPUT：書き込み INPUT　：読み取り APPEND：アペンド モードを指定するとシーケンシャルファイル、モードを指定しないとランダムアクセスファイルを開く。	OPEN "CRT：TEST"FOR OUTPUT AS #1
PRINT #ファイル番号, [式][セパレーター][式]	データを順にシーケンシャルファイルに書き込む。	PRINT #1, "ABC"
PRINT #ファイル番号, USING書式記号；式	指定した書式でデータを順にシーケンシャルファイルに書き込む。 ＊PRINT USINGの項参照。	PRINT #1, USING"&　　&" ,A$
INPUT #ファイル番号, 変数 [,変数] ……	データをシーケンシャルファイルから順に読み取り、変数に代入する。	INPUT #1,A,B,C
LINE INPUT #ファイル番号, 変数	254字までの文字列をシーケンシャルファイルから読み取り、変数に代入する。	LINE INPUT #1,A$
CLOSE [#] [ファイル番号] [,ファイル番号] …	ファイルを閉じる。	CLOSE #1,2
SAVE "デバイス名 [ファイル名]" ▣SAVE "MEM：ファイル名[.タイプ名]"	プログラムをアスキー形式でセーブする。(ディスク以外)	SAVE "CAS：PROGRAM"
⊟SAVE " [ドライブ名]ファイル名 [.タイプ名]" [,A]	プログラムをディスクにセーブする。 Aオプションをつけるとアスキー形式でセーブ、つけないと中間言語でセーブ。	SAVE "GAME1.BAS" SAVE "GAME2.ASC", A
LOAD "デバイス名 [ファイル名]"	アスキー形式のプログラムをロードする。(ディスク以外)	LOAD "CAS：PROGRAM"
▣⊟LOAD "[ドライブ名]ファイル名[.タイプ名]" [,R]	プログラムをディスクからロードする。 Rオプションをつけるとロードして実行する。	LOAD "GAME1.BAS",R
MERGE "デバイス名 [ファイル名]"	アスキー形式プログラムをロードし、メモリー上のプログラムと合併する。	MERGE "CAS：PROG2"
▣⊟MERGE "[ドライブ名] ファイル名 [.タイプ名]"	アスキー形式のプログラムをディスクからロードし、メモリー上のプログラムと合併する。	MERGE "GAME2.ASC"
BSAVE "デバイス名 [ファイル名]", 先頭番地, 終了番地 [, 実行開始番地]	指定したメモリー範囲の内容をセーブする。(ディスク以外)	BSAVE "CAS：GAME", &H3000, &H3FFF
⊟BSAVE "[ドライブ名]ファイル名[.タイプ名]", 先頭番地, 終了番地 { [, 実行開始番地] / [, S] }	メインメモリーまたはビデオRAMの内容をディスクにセーブする。 Sオプションをつけるとビデオ RAMの内容をセーブ、つけないとメインメモリーの内容をセーブ。	BSAVE "PROG.BIN",&HE000,&HE800 BSAVE "CHART",0,&H3FFF,S
BLOAD "デバイス名 [ファイル名]" [,R] [,オフセット]	機械語プログラムをロードする。(ディスク以外) Rをつけるとロードして実行。オフセットはロード時のメモリー番地に対するオフセット。	BLOAD "CAS：GAME" ,R
⊟BLOAD "[ドライブ名]ファイル名[.タイプ名]" { [,R] / [,S] } [,オフセット]	機械語プログラムをディスクからロードする。Rオプションをつけるとロードして実行する。Sオプションをつけると、ファイルのデータをビデオRAMにロードする。	BLOAD "PROG.BIN",R BLOAD "CHART" ,S

形　　式	機　　能	例
CSAVE "ファイル名" [, ボーレート]	プログラムをカセットに中間言語でセーブする。 ボーレート……1：1200ボー 2：2400ボー	CSAVE "STAR"
CLOAD ["ファイル名"]	カセットテープからプログラムをロードする。	CLOAD "STAR"
CLOAD? ["ファイル名"]	カセットテープにセーブしたプログラムとメモリー上のプログラムを比較照合する。	CLOAD? "STAR"
FIELD [#] ファイル番号, 文字長 AS 文字型変数 [, 文字長 AS 文字型変数] ……	ランダムアクセスファイルの1レコードの書式を定義する。	FIELD #1,12 AS NAM$, 14 AS TEL$
LSET　文字型変数＝文字式 RSET　文字型変数＝文字式	文字式の内容をレコード内に割り当てられた文字型変数に書き込む。 (LSETは左づめ, RSETは右づめ)	LSET TEL$=B$ RSET NAM$= "TOM"
PUT [#] ファイル番号 [, レコード番号]	レコードの内容をディスクのランダムアクセスファイルに書き込む。	PUT# 1,1
GET [#] ファイル番号 [, レコード番号]	ディスクのランダムアクセスファイルから1レコード読み込む。	GET #1,10

デバイス名……CAS：カセットテープ
CRT：テキスト画
GRP：グラフィック画
LPT：プリンター
MEM：メモリーディスク

A：, B：, C：, D：, E：, F：, G：, H： フロッピーディスクのドライブ名

漢　日本語入力に関するコマンド

形　　式	機　　能	例
CALL KANJI [漢字フォント]	日本語入力モードにする。	CALL KANJIØ
CALL ANK	日本語入力モードを終了する。	
CALL AKCNV(文字変数, X$)	全角文字に変換する。	CALL AKCNV(A$, "アイウエオ")
CALL JIS(文字変数, X$)	文字のJISコードを得る。	CALL JIS(A$, "漢")
CALL SJIS(文字変数, X$)	文字のシフトJISコードを得る。	CALL SJIS(A$, "漢")
CALL KACNV(文字変数, X$)	半角文字に変換する。	CALL KACNV(A$, "アイウエオ")
CALL KEXT(文字変数, X$, 機能)	全角文字または半角文字だけを抜き出す。	CALL KEXT(A$, "漢字ABC", 1)
CALL KINSTR(数値変数 [, N], X$, Y$)	X$のN文字目以降にあるY$の位置を調べる。	CALL KINSTR (A, "TOKYO東京", "K")
CALL KLEN(数値変数, X$ [, 機能])	文字データの長さを調べる。	CALL KLEN(A, "日本語")
CALL KMID(文字変数, X$, M [, N])	文字データの一部分を抜き出す。	CALL KMID(A$, "今日は", 3, 1)
CALL KNJ(文字変数, X$)	漢字コードから文字を得る。	CALL KNJ(A$, "3441")
CALL KTYPE(数値変数, X$, N)	文字が全角か半角かを調べる。	CALL KTYPE(A, "ABCあいう", 4)
CALL CLS	日本語入力モードのスクリーンのすべての表示を消去する。	
CALL PALETTE(パレット番号, 赤の輝度, 緑の輝度, 青の輝度)	日本語入力モードのとき，カラーパレットに色を割り当てる。	CALL PALETTE(15, 4, 4, 4)

メモリースイッチ機能に関するコマンド

形　　式	機　　能	例
SET ADJUST(X座標, Y座標)	画面の表示位置を変える。	SET ADJUST(−4,3)
SET BEEP 音色, 音量	ピッという音を変更する。	SET BEEP 2,3
SET TITLE [“タイトル”], [カラー]	初期画面に表示されるタイトルを指定する。	SET TITLE “SONY”
SET PROMPT “プロンプト”	入力待ちを知らせるプロンプト文を指定する。	SET PROMPT “Ready”
SET PASSWORD “パスワード”	パスワードを設定する。	SET PASSWORD “BASIC”
SET SCREEN	現在設定されているスクリーンの状態を初期状態とする。	

データの入出力に関するコマンド

形　　式	機　　能	例
DATA 定数[, 定数] [, 定数]……	READ文が読み取るデータを与える。	DATA3, 4, 5, 6, ABC, "C, D"
INPUT [“プロンプト文”;]変数[, 変数][, 変数]…	変数の値をキーボードから与える。	INPUT “A$=”;A$
LINE INPUT [“プロンプト文 ”;]変数	254字までの文字列を文字型変数にキーボードから与える。	LINE INPUT “C$=”;C$
[LET]変数=X	データを変数に代入する	LET A=A+5
MID$(X$, M[, N])=Y$	文字列X$のM番目からの文字を, Y$の先頭からN番目までの文字で置き換える。	MID$(A$, 2, 5)=B$
PRINT [式][セパレーター][式][セパレーター][式]…… または ?[式] [セパレーター] [式] [セパレーター] [式]……	データを画面に出力する。 セパレーターは;または,またはスペース	PRINT A;B;C
PRINT USING 書式記号;式 [, 式] …	指定した書式でデータを画面へ出力する。 書式記号 “!”　：最初の1字を出力。 “&n個のスペース&”：n+2字分出力。 “@”　：文字列全部を出力。 “#”　：数値データの表示けた数指定。 “+”　：数値データの前(後)に+、−をつける。 “−”　：負の数値データの後に−をつける。 “**”　：数値データの前を*で埋める。 “¥¥”　：数値データの前に¥をつける。 “**¥”　：数値データの直前に¥をつけ、それより前を*で埋める。 “,”　：小数点より左へ3けたごとに, を入れる。 “^^^^”：浮動小数点型で出力する。	10 A$=“ABCDEFG” 20 PRINT USING“!”;A$ 30 PRINT USING“&␣␣&”;A$ 40 PRINT USING“SSS@TTT”;A$ PRINT USING“###.##”;123.45, 10.5 PRINT USING“+###”;100, −200 PRINT USING“###−”;100, −200 PRINT USING“**###”;100, −200 PRINT USING“¥¥###”;100, −200 PRINT USING“**¥###”;10, −20 PRINT USING“####,.##”;1234.56 PRINT USING“#.##^^^^”;123.98
READ 変数[, 変数] [, 変数]……	DATA文中のデータを読み取る。	READ A%
RESTORE [行番号]	READ文が読み取るDATA文を指定する。	RESTORE 100
SWAP 変数, 変数	2つの変数の値を交換する。	SWAP A, B

フロッピーディスク，メモリーディスク管理に関するコマンド

形　　　式	機　　　能	例
CALL FORMAT	ディスクをフォーマットする	
FILES [“[ドライブ名] [ファイル名 [.タイプ名]]”]	ディスク上のファイル名を表示する。	FILES FILES “*.BAS”
KILL “[ドライブ名] ファイル名 [.タイプ名]”	ディスク上のファイルを消去する。	KILL “TEST.BAS”
NAME “[ドライブ名] 旧ファイル名 [.旧タイプ名]” AS “新ファイル名 [.新タイプ名]”	ディスク上のファイル名を変更する。	NAME “OLD.DAT” AS “NEW.DAT”
COPY “[ドライブ名1]ファイル名[.タイプ名]” [TO “[ドライブ名2] ファイル名 [.タイプ名]”]	ディスク上のファイルを同一ディスクまたは他のディスク上にコピーする。	COPY “ABC.BAS” TO “XYZ.BAS” COPY “A：ABC.BAS” TO “B：”
CALL MEMINI [(サイズ)]	メモリー内にメモリーディスク領域を設定し、初期化する。	CALL MEMINI (20000)
CALL MFILES	メモリーディスク内のファイル名を表示する	
CALL MKILL (“ファイル名 [.タイプ名]”)	メモリーディスク内のファイルを消去する。	CALL MKILL (“ADRS.DAT”)
CALL MNAME(“旧ファイル名[.旧タイプ名]”AS “新ファイル名 [.新タイプ名]”)	メモリーディスク内のファイル名を変更する。	CALL MFILES (“OLD.DAT” AS “NEW.DAT”)

割り込みに関するコマンド

形　　　式	機　　　能	例
ON KEY GOSUB 行番号 [,行番号] ……	ファンクションキーにより割り込みをかける。	ON KEY GOSUB 1000, 2000,3000
KEY (ファンクションキー番号) ON	ファンクションキーによる割り込みを有効にする。	KEY(1) ON
KEY (ファンクションキー番号) OFF	ファンクションキーによる割り込みを無効にする。	KEY(2) OFF
KEY (ファンクションキー番号) STOP	ファンクションによる割り込みを保留にする。	KEY(3) STOP
ON STRIG GOSUB 行番号 [,行番号] ……	コントローラーのボタンにより割り込みをかける。	ON STRIG GOSUB 1000,2000
STRIG (コントローラー番号) ON	コントローラーによる割り込みを有効にする。 コントローラー番号…… 0　：スペースバー 1,3　：コントローラーA 2,4　：コントローラーB	STRIG(1) ON
STRIG (コントローラー番号) OFF	コントローラーによる割り込みを無効にする。	STRIG(2) OFF
STRIG (コントローラー番号) STOP	コントローラーによる割り込みを保留する。	STRIG(0) STOP
ON STOP GOSUB 行番号	[CTRL]+[STOP] キーにより割り込みをかける。	ON STOP GOSUB 1000
STOP ON	[CTRL]+[STOP] キーによる割り込みを有効にする。	
STOP OFF	[CTRL]+[STOP] キーによる割り込みを無効にする。	
STOP STOP	[CTRL]+[STOP] キーによる割り込みを保留にする。	

形　　式	機　　能	例
ON SPRITE GOSUB 行番号	スプライトの衝突により割り込みをかける。	ON SPRITE GOSUB 1000
SPRITE ON	スプライトの衝突による割り込みを有効にする。	
SPRITE OFF	スプライトの衝突による割り込みを無効にする。	
SPRITE STOP	スプライトの衝突による割り込みを保留にする。	
ON INTERVAL＝インターバル GOSUB 行番号	一定間隔で割り込みをかける。割り込み間隔はインターバル×1/60秒	ON INTERVAL＝120 GOSUB 1000
INTERVAL ON	一定間隔の割り込みを有効にする。	
INTERVAL OFF	一定間隔の割り込みを無効にする。	
INTERVAL STOP	一定間隔の割り込みを保留にする。	

接続機器に関するコマンド

形　　式	機　　能	例
LPRINT [式][セパレーター][式][セパレーター][式]……	データをプリンターに出力する。	LPRINT A,B,C
LPRINT USING書式記号;式[セパレーター][式][セパレーター]……	データをプリンターに指定した書式で出力する。(書式は，PRINT USINGの項参照)	LPRINT USING "###";A,B
LLIST [先頭行番号][-][最終行番号]	プログラムリストを印字する。	LLIST 100−200
MOTOR[{ON / OFF}]	テープレコーダーのモーターをON/OFFする。	MOTOR OFF

内蔵タイマーに関するコマンド

形　　式	機　　能	例
SET DATE "年/月/日" [,A]	タイマーに日付けをセットする。	SET DATE "85/10/05"
GET DATE 文字変数 [, A]	現在の日付けを変数に代入する。	GET DATE D$
SET TIME "時：分：秒" [, A]	タイマーに時刻をセットする。	SET TIME "14：05：00"
GET TIME 文字変数 [,A]	現在の時刻を変数に代入する。	GET TIME T$

エラー処理に関するコマンド

形　　式	機　　能	例
ERROR エラー番号	指定したエラー番号のエラーを発生させる。エラー番号を定義する。	ERROR 3 IF A>100 THEN ERROR 250
ON ERROR GOTO 行番号	エラーが発生したら、指定した行番号へ実行を移す。	ON ERROR GOTO 1000
RESUME[{0 / NEXT / 行番号}]	エラー処理ルーチンの実行後、メインプログラムに実行を戻す。	RESUME 10

機械語サブルーチンに関するコマンド

形　　式	機　　能	例
DEFUSR [X]＝開始番地	機械語サブルーチンの開始番号を指定する。	DEFUSR0＝53248
POKE 番地，式	メモリーにデータを書き込む。	POKE &HA400,&HFF

I/Oポート，メモリーに関するコマンド

形　　式	機　　能	例
OUT　ポート番号，式	I/Oポートに1バイトのデータを出力する。	OUT &H90,3
WAIT　ポート番号，式1[,式2]	I/Oポートからの入力データがある値になるまでプログラムの実行を中断する。	WAIT &H90,255
VPOKE　アドレス,式	ビデオRAMに1バイトのデータを書き込む。	VPOKE 263,01

拡張コマンドに関するコマンド

形　　式	機　　能	例
CALL　拡張コマンド[(引数，引数……)] または _拡張コマンド　[(引数，引数……)]	ROMカートリッジの拡張コマンドを呼びだす。	CALL SUB

DOSに制御を移すコマンド

形　　式	機　　能	例
🖫CALL SYSTEM	MSX-DOSに制御を移す。	

エラーメッセージ

1：NEXT without FOR　(NEXT文に対応するFOR文がない)
2：Syntax error　(文法上の誤り)
3：RETURN without GOSUB　(RETURN文に対応するGOSUB文がない)
4：Out of DATA　(データが足りない)
5：Illegal function call　(関数やコマンド中の指定が不適当)
6：Overflow　(オーバーフロー)
7：Out of memory　(メモリーが足りない)
8：Undefined line number　(定義されていない行番号を指定した)
9：Subscript out of range　(配列の添字が定義した範囲外)
10：Redimensioned array　(DIM文で配列を再定義しようとした)
11：Division by zero　(ゼロで割った)
12：Illegal direct　(直接コマンドモードでは使えないコマンドを使った)
13：Type mismatch　(型が合わない)
14：Out of string space　(文字変数の領域が足りない)
15：String too long　(文字列が長すぎる)
16：String formula too complex　(文字式が複雑すぎる)
17：Can't CONTINUE　(続行不可能)
18：Undefined user function　(DEF FN文で定義されていない関数を使った)
19：Device I/O error　(接続機器にエラーがあった)
20：Verify error　(カセットテープ上のプログラムとメモリー上のプログラムが異なる)
21：No RESUME　(ON ERROR文に対応するRESUME文がない)
22：RESUME without error　(RESUME文に対応するON ERROR文がない)
23：Unprintable error　(エラーメッセージのないエラーが発生した)
24：Missing operand　(オペランドがない)
25：Line buffer overflow　(入力されたプログラムがバッファサイズを越えた)
50：🖫FIELD overflow　(FIELD文での領域の指定が、レコードの長さを越えた)
51：Internal error　(メモリーやテキストの内容の状態が正常でない)
52：Bad file number　(ファイル番号の誤り)
53：▥🖫File not found　(指定したファイルが存在しない)
54：▥🖫File already open　(すでに開かれているファイルにKILLコマンドを使おうとした)
54：File already open　(すでに開かれたファイルをOPENしようとした)
55：Input past end　(最後のデータの読み取りが終わっているのにさらに読もうとしている)
56：Bad file name　(ファイル名の指定の誤り)
57：Direct statement in file　(ファイルをロード中直接コマンドモードのコマンドが与えられた)
58：Sequential I/O only　(シーケンシャルファイルにGET文、PUT文を使おうとした)
59：File not OPEN　(ファイルが開かれていない)
60：🖫Bad FAT　(ディスクがフォーマットされていない)
61：Bad file mode　(シーケンシャルファイル、ランダムアクセスファイルのコマンド、関数の使い方を間違えた)
62：🖫Bad drive name　(使われていないドライブ名を指定した)
63：🖫Bad sector number　(PUT文、GET文で指定したレコード番号が0、または32767より大きい)
64：▥🖫File still open　(ファイルが閉じられていない)
65：File already exists　(NAME、CALL MNAME文で新しいファイル名として指定したファイルがすでに存在する)
66：🖫Disk full　(ディスクの残り容量がなくなった)
66：▥[RAM] disk full　(メモリーディスクの残り容量がなくなった)
67：Too many files　(ファイルの数が255を越えた)
68：🖫Disk write protected　(書き込み禁止のディスクに対して書き込みを行った)
69：🖫Disk I/O error　(ディスクとの入出力時に、回復不可能なエラーが生じた)
70：🖫Disk offline　(ディスクドライブがつながっていない)
70：▥[RAM] disk offline　(CALL MEMINIを実行しないでメモリーディスクを使おうとした)
71：🖫Rename across disk　(異なるディスクドライブ間でNAME文を実行しようとした。)

関数

数値型関数

ABS(X)	絶対値を返す。
ATN(X)	アークタンジェントを返す。
CDBL(X)	倍精度のデータにして返す。
CINT(X)	整数型に変換する。(−32768≦X<32768)
COS(X)	コサインを返す。
CSNG(X)	単精度のデータにして返す。
ERL	エラーが起こった行の行番号を返す。
ERR	発生したエラーのエラー番号を返す。
EXP(X)	e^xを返す。
FIX(X)	Xの整数部分を返す。
INT(X)	X未満の最大の整数を返す。
LOG(X)	自然対数を返す。
RND(X)	0以上1未満の乱数を返す。
SGN(X)	X>0なら1, X=0なら0, X<0なら−1を返す。
SIN(X)	サインを返す。
SQR(X)	平方根を返す。
TAN(X)	タンジェントを返す。

文字型関数

LEFT$(X$, N)	X$の左からN文字を返す。
MID$(X$, M[, N])	X$のM文字目以降のN文字を返す。
RIGHT$(X$, N)	X$の右からN文字を返す。
SPACE$(N)	N文字分のスペースを文字データとして返す。
STRING$(N, J)	Jを文字コードとする文字をN字分、文字データとして返す。
STRING$(N, X$)	X$の最初の文字をN字分, 文字データとして返す。
TAB(N)	N文字目にカーソルを移動する。
SPC(N)	N文字分のスペースを出力する。

数値型↔文字型変換に関する関数

ASC(X$)	X$の先頭文字のキャラクターコードを返す。
BIN$(X)	Xの2進表示を文字型データとして返す。(−32768≦X<65536)
CHR$(X)	キャラクターコードがXである文字を返す。
HEX$(X)	Xの16進表示を文字型データとして返す。(−32768≦X<65536)
INSTR([N,]X$, Y$)	X$のN文字目以降にあるY$の位置を返す。
LEN(X$)	X$の文字数を返す。
OCT$(X)	Xの8進表示を文字型データとして返す。(−32768≦X<65536)
STR$(X)	文字型に変換する。
VAL(X$)	数値型に変換する。
CVI, CVS, CVD	ランダムアクセスファイル内の文字型データを数字型に変換する。
MKI$, MKS$, MKD$	ランダムアクセスファイルに書き込む数値を文字型に変換する。

その他の関数

PLAY(N)	音楽演奏中かどうかを調べる。 N=1, 2, 3のとき：演奏中なら−1を、そうでなければ0を返す。 N=0のとき：すべてのミュージックサブコマンドの状態(−1または0)のORをとって返す。

データ入力に関する関数

スクリーンから

CSRLIN	カーソルの位置のY座標を返す。
POS(X)	カーソルの位置のX座標を返す。
POINT(X, Y)	(X, Y)の位置の点のカラーコードを返す。

データファイルから

EOF(ファイル番号)	ファイルの最後のデータが読み取られたら−1, それ以外では0を返す。
INPUT$(N, [#]ファイル番号)	ファイルからN文字入力して返す。
LOF(ファイル番号)	ファイルの長さ(バイト)を返す。
LOC(ファイル番号)	ファイルの中の現在位置を返す。

プリンターから

LPOS(X)	プリンターバッファ内のプリントヘッドの位置を返す。

メモリーから

FRE(0)	メモリーの未使用領域を返す。
FRE("")	文字列領域の未使用部分を返す。
PEEK(番地)	メモリー中のデータを読み取る。
VARPTR(変数)	変数の格納されている先頭番地を返す。
VARPTR(#ファイル番号)	ファイルが割り当てられているファイルコントロールブロックの先頭番地を返す。
VPEEK(番地)	ビデオRAMの中のデータを読み取る。

キーボードから

INKEY$	押されたキーの文字を返す。
INPUT$(X)	キーボードからX字入力する。

ディスクから

DSKF(ドライブ番号)	ディスクの残り容量をクラスター単位で返す。

I/Oポートから

INP(ポート番号)	I/Oポートからデータを入力する。

機械語サブルーチンから

USR{0 〜 9}(X)	ユーザーサブルーチンからの値を返す。

ジョイスティック, パドル, タッチパッドから

STICK(N)	ジョイスティックの方向を返す。(N=0はカーソル移動キー)(中央：0、上：1、右上：2、右：3、右下：4、下：5、左下：6、左：7、左上：8)
STRIG(N)	コントローラーのボタンが押されていれば−1を、押されていなければ0を返す。(N=0はスペースバー)
PDL(N)	パドルの状態を返す。
PAD(N)	コントローラーの状態を返す。 N=0または4のとき　タッチパッドが触れられていれば−1、触れられていなければ0 N=1または5のとき　タッチパッドに触れた位置のX座標 N=2または6のとき　タッチパッドに触れた位置のY座標 N=3または7のとき　タッチパッドのスイッチが押されていれば−1、そうでなければ0 N=8のとき　ライトペンのデータが有効なら−1, 無効なら0 N=9のとき　ライトペンのX座標 N=10のとき　ライトペンのY座標 N=11のとき　ライトペンのスイッチが押されていれば−1, そうでなければ0 N=12または16のとき　マウス、トラックボールに対する入力要求（必ず−1を返す。） N=13または17のとき　マウス、トラックボールのX座標 N=14または18のとき　マウス、トラックボールのY座標 N=15または19のとき　必ず0を返す。

定数と変数

定数	文字型	0～255文字の文字列 (クォーテーションで囲む)
	整数型	－32768～＋32767
	固定小数点型	小数点を含む数
	浮動小数点型	有効数字6けた（単精度）または14けた（倍精度） 指数部　－64～＋62
	16進型式	&Hを前につける
	8進型式	&Oまたは&を前につける
	2進型式	&Bを前につける

変数	変数名	始めの2文字のみ有効
	型宣言文字	変数名の後につける %：整数型　#：倍精度 !：単精度　$：文字型

特別な変数

TIME……内蔵タイマーの値を保持する。書き換え可。

SPRITE$（スプライト番号）……スプライトパターンを保持する。

［例］ SPRITE$(1)＝CHR$(&H18)＋CHR$(&H3C)＋CHR$(&H7E)＋CHR$(&HFF)＋CHR$(&H18)＋CHR$($H18)＋CHR$(&H18)＋CHR$(&H18)

●VDPに関する特別なコマンド、関数

BASE(式)……VDPテーブルのベースアドレスの読み出し、書き込みに使用する。

VDP(数値)……VDPレジスターの内容の読み出し、書き込みに使用する。

カラーコード

コード	色	コード	色
0	透　明	8	赤
1	黒	9	淡い赤
2	緑	10	濃い黄
3	淡い緑	11	淡い黄
4	濃い青	12	濃い緑
5	淡い青	13	マゼンタ
6	濃い赤	14	灰
7	空　色	15	白

演算記号

算術演算記号	^	累乗	MOD	整数剰余
	－	符号を変える	＋,－	加算，減算
	*,/	乗算、除算		(上から優先の順)
	¥	整数除算		
関係演算記号	< >＝	大小比較		
論理演算記号	NOT	否定	XOR	排他的論理和
	AND	論理積	EQV	排他的論理和の否定
	OR	論理和	IMP	含意

3-786-730-**01** (1)

MSX-BASIC Version 3.0

解説書

ソニー株式会社　〒141 東京都品川区品川6-7-35

お問い合わせはお客様相談センターへ
●東京(03)448-3311 ●大阪(06)251-5111 ●名古屋(052)232-2611

Printed in Japan